REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
14-12-2005

2. REPORT TYPE
Final Report

3. DATES COVERED *(From – To)*
01-Dec-00 - 01-Dec-05

4. TITLE AND SUBTITLE

Mathematical Basis of Knowledge Discovery and Autonomous Intelligent Architectures - Technology for the Creation of Virtual objects in the Real World

5a. CONTRACT NUMBER
ISTC Registration No: 1993p

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

B.V.Sokolov, F.M.Kulakov

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

St. Petersburg Institute For Informatics & Automation of the Russian Academy of Sciences
39, 14th Liniya
St. Petersburg 199178
Russia

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0014

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ISTC 00-7031-6

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Global awareness (GA) entails the acquisition of data from local to global levels, appropriate fusing of the data, and presentation of that data as useful information. This data will then be fused to fully describe situations of interest such as large transportation systems and complex communication systems. This project specifically aims at developing the mathematical basis, architecture and software techniques implementing particular new technologies to support Global Awareness and comprises six main tasks. Task 6 was:

6. Technology for Creation of Virtual Object in the Real World

15. SUBJECT TERMS

EOARD, Mathematical And Computer Sciences, Computer Programming and Software

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18, NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			PAUL LOSIEWICZ, Ph. D.
UNCLAS	UNCLAS	UNCLAS	UL	140	**19b. TELEPHONE NUMBER** *(Include area code)* +44 20 7514 4474

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39-18

ST. PETERSBURG INSTITUTE FOR INFORMATICS AND AUTOMATION

EUROPEAN OFFICE OF AEROSPACE RESEARCH AND DEVELOPMENT (EOARD)

Technology for the Creation of Virtual objects in the Real World

Final Report

Project # 1992 P

Task 6

Project Manager
Dr.Sci., Professor
B.V.Sokolov

Submanager
Dr.Sci., Professor
F.M.Kulakov

St. Petersburg

November 2003

Contents

Foreword

The submitted report presents a summary of results of the work over the task "Technology for Creation of Virtual Object in the Real World".

This task corresponds to Task 6 of the Partnership Project # 1992p.

The main objectives of the task are:

1. To develop advanced multimedia technology for creation ("immersion") of virtual objects in the real world (environment) providing visual and tactile/kinaesthetic effects of "interaction" between man's hands and virtual object.
2. To verify and to validate the developed Technology by testing with hard&software complex designed for these aims.

The Report gives an account in detail the approaches to realization of both visual and tactile-kinaesthetic effects in "immersion" of virtual objects in real scene when virtual objects are solid bodies or a telecontrolled manipulator. It describes the developed methods, algorithms and software means realizing the proposed approaches and gives characteristics and design of the developed and fabricated Hard&Software Complex Facility which the developed technology was tested on. It gives the results of the experimental studies confirming validity of the proposed technology, briefly sketches modifications to this technology needed for rising the realizm of visual effect of "immersion" for a case of moving virtual and real objects.

Chapter 1. Introduction

One of problem in the virtual reality is one immersing virtual objects in the real environment providing effects of visual and tactile-kinaesthetic perception. Thus, the problem has two aspects: visual and tactile-kinaesthetic.

The visual aspect, in its turn, has two variants depending on whether man sees real environment, a virtual object is to be immersed to, with a video camera and Helmet Mounted Displays (Fig.1.1) or immediately through an optical system (Fig.1.2) [1]. The first variant has title "Video see through HMD", second - "Optical see-through HMD". When a real world zone is at a considerable distance from observer the first option is the only possible. The second option is possible only in a case of the zone's being close by.

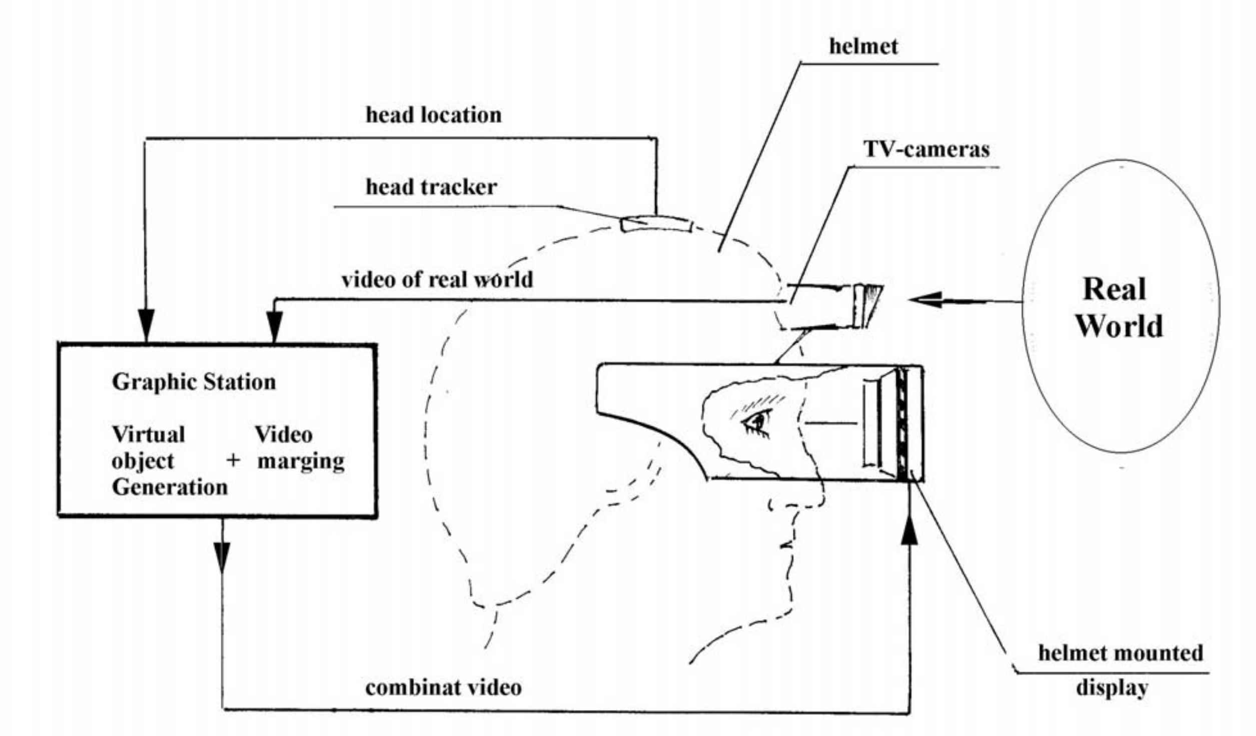


Fig.1.1 A variant of immersing virtual bodies in real environment seen by a TV camera.

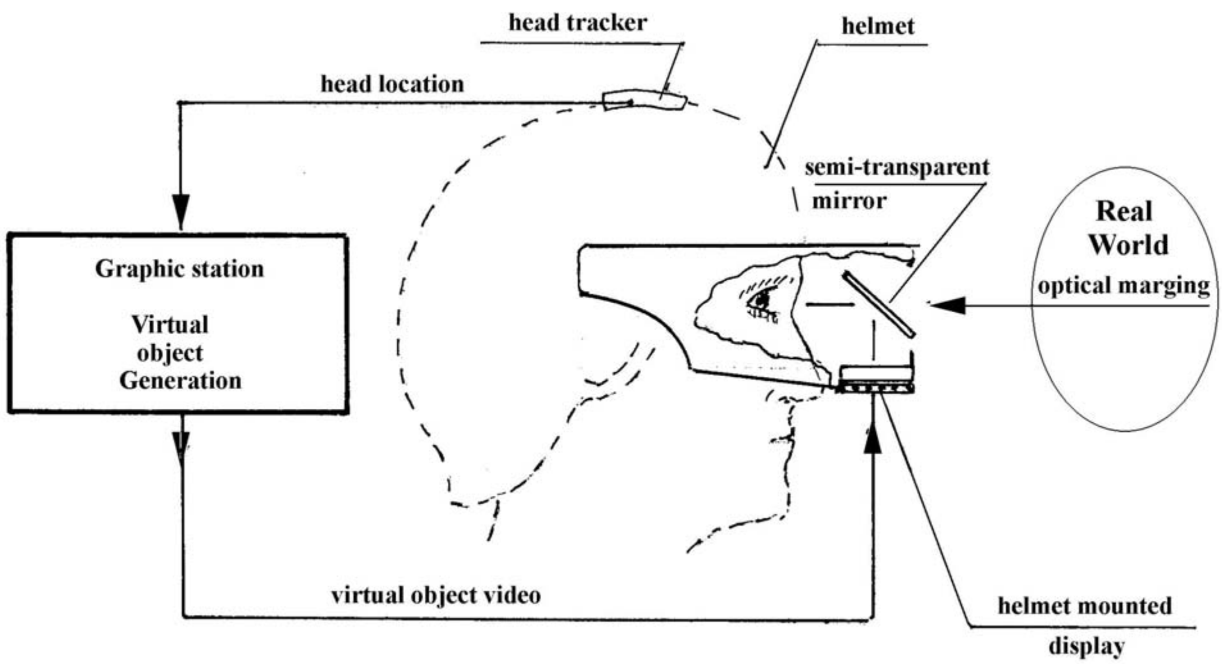


Fig.1.2 A variant of immersing virtual bodies in real environment seen through an optic system.

A visual effect of immersion (first variant) implies observer's being able to see:

(1) stereo pairs of images of environmental objects obtained with coupled video cameras mounted in the work scene which may be at a considerable distance from observer;
(2) stereo pairs of images of computer-synthesized objects perceived as real amidst real environmental objects;
(3) stereo pairs of images of human hands obtained with special coupled video cameras mounted in the room observer sits in which are perceived as being amidst real environmental objects.

A tactile-kinaesthetic aspect implies that man is able to perceive with the help of his hands shape, mass, weight and texture of bodies and muscular strain opposing to friction having the illusion that they are real bodies. In a specific case, when a virtual body is a robot-manipulator telecontrolled in master-slave mode providing, so-called, effort reflection [2], the abovementioned third condition for realization of the visual effect, precisely, obtaining the of human hands image, is absent.

As for realizing the tactile-kinaesthetic effect, in this case of man moving the control handle must perceive the same reaction forces as arise while master-slave-controlled of real manipulator [3].

For the second variant the visual effect of immersion implies the same as for the first one for one exception: images of environment and hands are obtained not with the help of cameras but with an optic system. The tactile-kinaesthetic effect of the first variant now reduces to only the tactile one in “interaction” of hands with virtual bodies.

The problem to consider relates to technique of, so-called, Augmented Reality [4-7] which is a quickly developing trend of Virtual Reality. Virtual object in this case is augmented to reality what makes the term.

Many companies and work teams are working over these problems. A number of successful decisions have found application in surgery, projection, industry and, also, in master slave and supervisory telecontrol of mobile objects, space and submarine robot-manipulator as such [8-16]. In the latter case virtual objects are virtual robots [17-21]. They are used as man-controlled models of robots for preliminary check of real robot’s action.

Yet, despite of the success attained in this field, researches need to be continued.

The main objectives of the considered task are:

1. To develop advanced multimedia technology for creation (immersion) of virtual object in the real world (environment) providing visual effect of perception and also tactile/kinaesthetic of effect "interaction" between man's hands and virtual object.
2. To verify and to validate the developed Technology by testing with hard&software complex tailored for these aims.

The developed technology must provide a perception of virtual objects in real environment more realistic for man and also to make Augmented Reality systems less expensive and extend the field of their application.

Chapter 2. An approach to realization of the visual aspect of immersion

The visual effect of virtual body's immersion, or in other term, augmentation to the real environment for a case when man uses a video camera (the first variant of immersion) implies the following. First, eyes of man are brought to a distant place of interest (work scene) using, for providing the stereo effect, two coupled cameras placed in the scene which generate and transmit the work scene image to two displays (for left and right eyes) mounted in operator's helmet. Moreover, operator/s hands are imaged with two additional head-borne cameras mounted near his eyes and transmitted to the same two helmet-mounted displays. Second, a pair of virtual object's images are synthesized with the help of computer and transmitted to the same displays, a pair of images of the work scene's geometric (topographic) model in the work scene's coordinate system obtained with the help of two virtual cameras with the same position/orientation relative to the model work scene as real ones to the work scene. Plus, images are generated of geometric model of human hands in their real current configuration obtained with the help of two virtual cameras in the same position/orientation as the real helmet-mounted cameras near eyes. Third, aspect and scale of images of virtual object, models of hands and work scene conform to the position/orientation of the head or, more precisely, eyes and change in real time following current head's position/orientation. Owing to that, in the ideal case model images of work scene and hands are registered on the helmet displays with their real images at any position/orientation of operator's head and have the same aspect and scale.

Fourth, the effect of screening is realized, viz. parts of real images which are behind virtual body or hands and hence invisible to operator are substituted for the screening parts of images of virtual body or hands. Fifth, position and orientation of virtual object are controlled by man and the same for virtual hands, adequate to real current situation.

For realizing a visual effect of "immersion" we propose a soft&hardware complex whose structure is shown in Fig.2.1 and Fig.2.2. Fig.2.1 corresponds to a case when virtual object is a telecontrolled manipulator and Fig.2.2 – to a case of arbitrary virtual body moved with hands.

This complex comprises the following functional units:

Unit 1. A stereo pair of movable TV cameras for making image of remote real environment (work scene). These cameras are mounted on and moved by a special robot-like device usually utilized for, so-called, active anthropomorphic vision [23] and serve for tracking position/orientation of operator's head.

Unit 2. A system for control of position/orientation of mobile TV cameras.

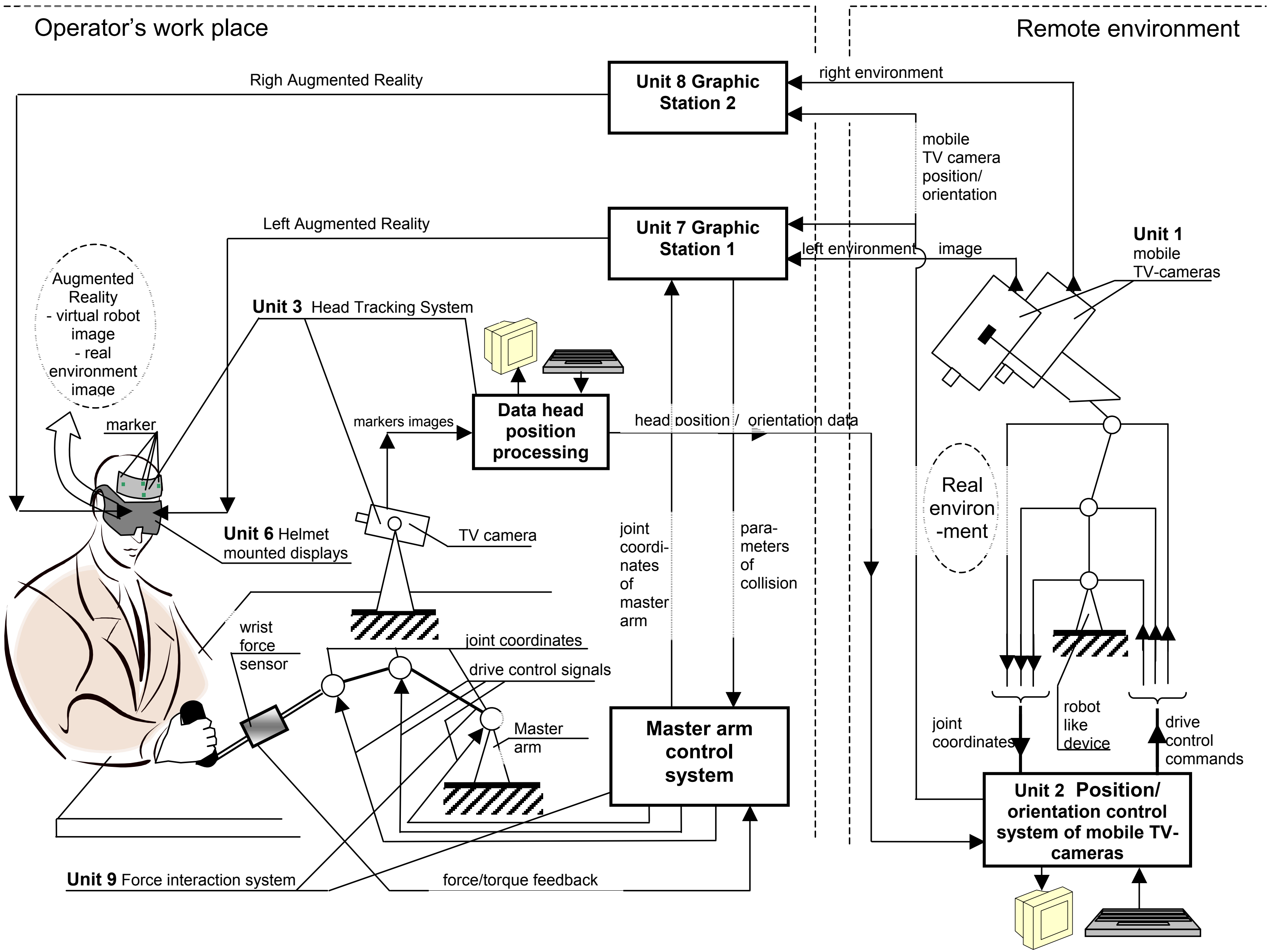


Fig.2.1 HSC Prototype structure with virtual manipulator

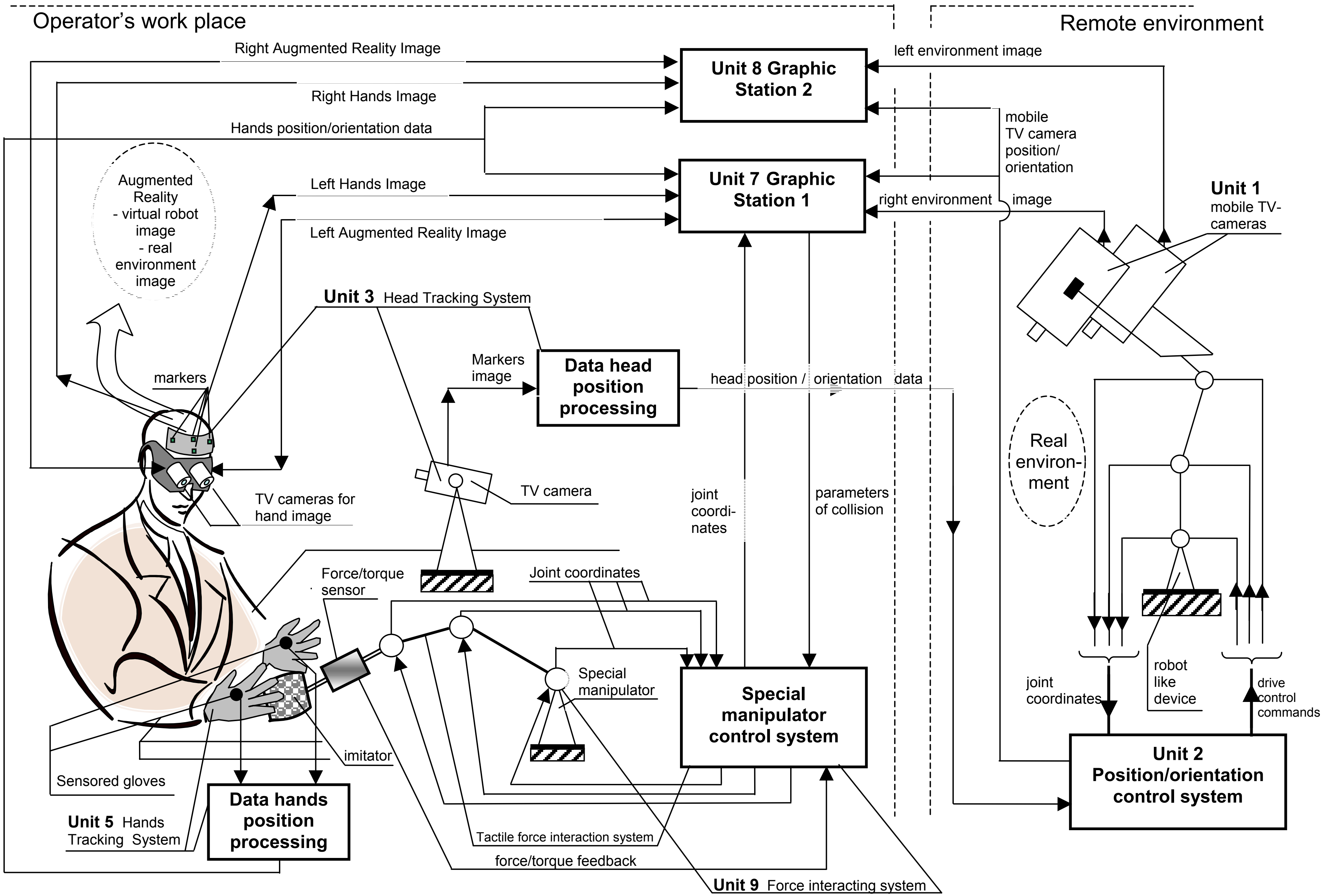


Fig. 2.2 HSC Prototype structure with virtual body

Unit 3. A system for tracking position/orientation of operator's head (Head Tracker System – HTS) incorporating a sensor for acquisition of primary data on position/orientation of head and a system processing these data for computing three linear and three angular current coordinates; these coordinates are target ones for the robot-like device's control system (Unit 2) with whose help the pair of TV cameras track position/orientation of head. The sensor may be based on any principles: electromagnetic, inertial etc. In our case it is optic-television principle which is presented in Fig. and Fig. based on using TV images of four reference marks on operator's helmet for calculation of head's position/orientation coordinates. Position of TV camera taking images of the marks is fixed in the coordinate system in which position/orientation of head is determined.

Unit 4. Special coupled video cameras mounted on head near eyes for taking stereo images of hands.

Unit 5. A system for tracking position and orientation of all mobile parts in hands and arms of man (forearms, palms, phalanxes of fingers) appearing as elements of their geometric (topographic) models. In a case when virtual object is a manipulator this unit is not used but now we need data on current coordinates of its joints obtained from position sensors on joints of the master arm (see Unit 1) which joint coordinates are tracked by the virtual manipulator.

Unit 6. Two displays built in the helmet (left and right) with a special optical device for presenting alternately to left and right eye the following images: that of environment obtained by means of stereo pair of mobile TV cameras borne by the robot-like device, that of hands obtained with the couple of video cameras mounted on head near eyes, that of virtual body obtained by means of computer synthesis.

Unit 7. Graphic station 1 whose SW's main purpose is to provide the visual effect of virtual body's immersion in real environment which requires the following functions:

a) Entering and digitizing work scene's video image obtained with the help of the left camera for observation of work scene;
b) Entering current data on coordinates representing position/orientation of all mobile elements in geometric models of arms and hands or, in a case when virtual body is a manipulator, current joint coordinates of the master arm representing current configuration of the virtual manipulator tracking the master arm configuration.
c) Generating geometric models of virtual object and work scene. In a case when the virtual object is moved with man's hands, one needs, additionally, to generate

geometric models of arms and hands on the base of previously entered data characterizing these models and also data on coordinates of mobile parts in arms and hands. If the virtual body is a manipulator one needs, additionally, data on current joint coordinates of the master arm.

d) Entering and digitizing TV images of man's hands obtained with the help of the left special TV camera on head.

e) Entering current coordinates on position/orientation of the left mobile TV camera for observation of work scene. In a case when virtual body is moved with hands one needs also current coordinates on position/orientation of the left special TV camera on head purposed for watching hands. It is noteworthy that position/orientation of both TV camera for observation of work scene and the special TV cameras on head, ideally, must copy those for eyes, deviations are determined by means of previous calibration and they are corrected for in generating virtual images of objects, hands and environment.

f) Generating the augmented reality, viz. obtaining images of real work scene and hands augmented with virtual object's image, their aspect and scale conforming to position/orientation of the left TV cameras for observation of work scene and hands. This process includes generation of model images of work scene and hands when virtual object is moved with hands.

g) Localizing 2D zones, unscreened for observer, of fragments of the geometrical model's images of work scene, virtual body and hands (when the body is moved with hands, see Fig.2.3) and substituting them for corresponding fragments of real environment and hands taken with video cameras. The fragments' boundaries of model images and real images must be precisely registered while substituting.

h) Outputting the "augmented reality" (that is real images of work scene and arms and hands augmented with image of virtual body) to the left helmet-mounted display.

Besides the listed functions the graphic station's SW must provide a part of functions for realizing the tactile-kinaesthetic aspect in immersion of virtual body in real environment and namely:

i. Detecting a fact of hand's contact with virtual body moved with them or virtual manipulator's contact with an environmental object if the virtual object is the manipulator.

ii. Generating data needed for computing expected values of force and torque vectors of virtual manipulator's interacting with environmental objects.

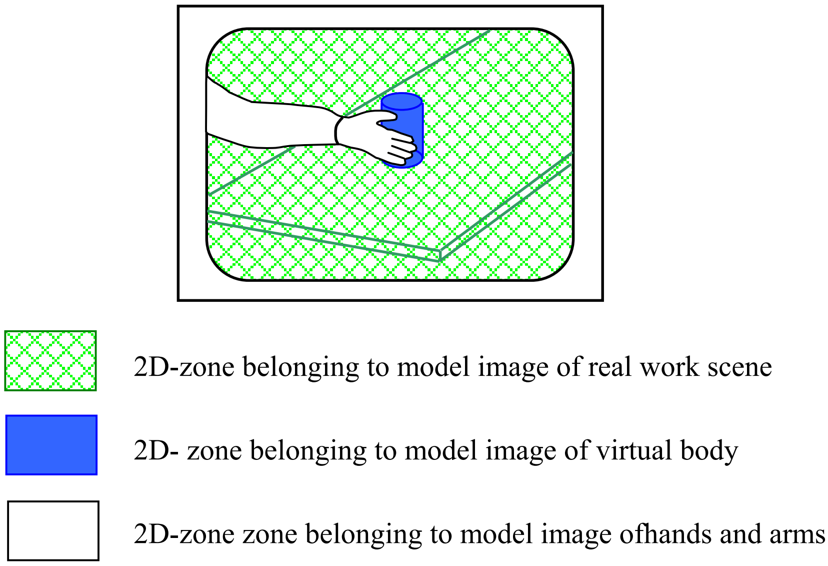


Fig. 2.3 Localizing 2D zones belonging to model images of work scene, virtual body and arms and hands

Unit 8. Graphic station 2. Its SW realizes the same functions as that of GS1. The difference lies in generating “augmented reality” for right eye and outputting it to the right helmet-mounted display. Using two GS1 for generating real work scene image augmented with model stereo image of virtual object is justified by the necessity to reduce time expended on synthesis of one image frame in order to attain a frame frequency enough for viewing an augmented image rendered without any visible jumps. This property provides a continuity of visual perception. Graphic Station 2 does not generate data for simulating tactile-kinaesthetic effect of immersion, it is done by GS1.

Unit 9. A special device for simulating the tactile- kinaesthetic effect of immersion. When virtual body is a manipulator it comprises:

- master arm with 6 joints provided with 6 drives and 6D wrist sensor measuring vectors of force and torque applied to the handle on the arm's end;
- device for the drive’s control, the SW’s main function is computing force and torque vectors of virtual manipulator’s interaction with an obstacle or other objects of real work scene.

If virtual body is an arbitrary body moved with hands of man this special device comprises:

- robot-like mechanism with 6 DOF provided with 6 drives and gripper handling exchangeable mock-ups of virtual body and 6D wrist sensor measuring vectors of force and torque applied to the mock-up by hand;
- device for control of robot-like mechanism's drives whose SW generates the controls ensuring such mock-up’s movement in space as if it were a real body having specified mass and inertia matrix.

Data on Unit 9 in more detail will be given in section 4.1.

The described approach to realizing the visual aspect of virtual body’s immersion in real environment requires solving a number of problems. The most critical and complex of them are two. The first one is the problem of precise registration of unscreened fragments of the real environment images and hands with corresponding fragments of geometrical model images and keeping this precise registration in real time with changing positions of man-observer and virtual object.

The second one is the problem of providing visual perception continuity of Augmented Reality image. Data processing for making a frame of “augmented reality” requires a huge mass of computation what makes difficult to provide frame frequiency excluding any visible jumps. They are possible as the operator’s hands and virtual objects are mobile. Moreover, aspects and scales are to be dynamically adjusted to possible movements of observer.

The major' part of computation is made for entering and digitizing TV images of real environment and hands, generating their geometric models and model images augmented with virtual object image, providing the effect of screening and, finally, outputting these images to the helmet displays. It is these computations, as was noted above, are to be made by the graphic stations.

Using for that two graphic stations, in place of one, simply and naturally divides the computational flow in two parallel branches and so provides a simple solution to the problem of stereo vision.

Taking in consideration that a mass of computation depends in considerable degree from a degree of detail in geometric models, one must choose a reasonable compromise between elaborateness of details in geometric models and time expended on that.

The complexity of problem of registration is determined by a high resolution of the eye retina which is 0,5 ang. min. Using a video camera for observating real environment results for the best video systems in 2,5 ang. min. or on the helmet display in size of one pixel. Thus, even one pixel of inaccuracy in registration will be perceived by eye and spoil realism of picture.

Errors of registration divide to two kinds: static and dynamic. Static errors are those which exist even when both observer's and observed object's positions/orientations are unchanged. These are errors caused by inaccuracy in determining position/orientation of TV cameras for work scene observation, leading to misalignment of real and virtual observers, and errors caused by, e.g., various kinds of nonliniarities in taking video image etc. Dynamic errors are those which show themselves when observer or observed object move. Dynamic errors make a major part of total error of registration if one uses helmet displays for Augmented Reality systems.

Unlike the above first variant of virtual body's immersion, in realizing the second variant of immersion with utilization of optic system one needs a special optic system for obtaining real images of work scene and hands, one which forms the stereo pair of scene images on a semitransparent screen.

The stereo pair of virtual images of required aspect and scale must be generated with SW of graphic stations. These images should appear on the same semitransparent screen to be amidst objects of work scene with hands to be perceived as real. For such immersion it is expedient to use special displays built in the helmet. The latter has optical systems for each eye projecting in appropriate scale the stereo pair of real images of environment and hands registered with a virtual body's image.

Chapter 3. An approach to realization of the tactile-kinaesthetic aspect of virtual object’s immersion in real environment

3.1 An approach to simulating the tactile-kinaesthetic interaction of hands with virtual body

In a case when virtual object is arbitrary solid body the simulation of the tactile-kinaesthetic interaction of hands with it means that, at zero distances between any part of real hand and the virtual body, this part must feel a tactile-kinaesthetic reaction as if it were real body of certain mass, inertia matrix and texture and the virtual body would move so as the real body, having the same inertial parameters would do under action of reaction forces arising at the impact.

A row of technical means are known to-day for realizing the tactile-kinaesthetic effect in the immersion, e.g. Cyber Force, Cyber Grasp, Cyber Glove produced by Virtual Technologies Company. Means proposed in the paper will, expectably, make possible a more realistic simulation of interaction. These means are a special 6-DOF manipulator and a system for control of its drives providing kinaesthetic simulation of hand’s interacting with virtual body. In Fig.2.2 these means are incorporated in Unit 9. The manipulator, instead of gripper, has mounted on its end exchangeable mock-ups of employed virtual bodies having similar shapes and textures but made of some light material to exclude effect of mock-up’s mass-inertial characteristics on the manipulator’s movement.

The manipulator is provided with a wrist force&torque sensor which measures all 6 components of resultant vector of force and torque arising from action of hand. This vector is, obviously, equal and opposite in sign to the vector of reaction acting on hand thus simulating a specified interaction.

The manipulator may operate in the following two alternative modes.

The thirst one is active when hand is at some distance from a body. The manipulator here keeps a mock-up in a position adequate to that of virtual body. This mode implies free motion of hand in space with the goal to grasp or push the body.

The second mode is active when distance between virtual body and hand becomes null. As a matter of course, the distance between the mock-up and hand is also null. This mode is purposed for simulating tactile and kinaesthetic interaction which would be if hand pushed or grasped and moved a real body with certain shape, size, inertia, weight and friction.

In either mode man should see neither manipulator nor mock-up. He should see by means of helmet displays that he grasps real body and feel real texture, weight, inertia friction resistance of body.

The manipulator may appear in FOV of cameras imaging operator's hands. But it may be easily detected on scene's image being of some specific, e.g. black, color. If so, it may be made invisible on computer scene's image by changing its color to that of background.

Realizing this kind of interaction requires control of manipulator's drives with a law adequate to that of real body with the same mass-inertial characteristics, weight and friction acted on with forces applied to mock-up by hand.

If hand apply in point A of mock-up a resultant force F and torque M (Fig.3.1) there exist the following conditions of equilibrium:

$$F + G_M + F_s = P_d,$$

$$M + r_a \times F + r_{cm} \times G_M + M_s = M_d,$$

where F_s and M_s are force and torque of reaction acting on mock-up from the sensor and measured with it; r_a is vector to point A where force F is applied; r_{cm} is vector to mock-up's center of gravity in the sensor's frame of coordinates; G_M is mock-up's gravity vector.

Then, taking in account relative small values of dynamic force P_d and torqe M_d acting on mock-up, due to insignificantly small mock-up's mass and inertia, as compared with real body, the following expressions are just for force F_s and M_s:

$$F_s = -F - G_M, \qquad M_s = -M - r_a \times F - r_{cm} \times G_M .$$

As a matter of course, knowing these sensor values and mock-up's weight and center-of-gravity position r_c in the sensor's frame of coordinates, one may find force F and total torque M_0 applied to the body.

Taking that it is possible to provide $r_c = r_{cm}$, we get finally:

$$M_0 = -M_s + r_c \times F_s.$$

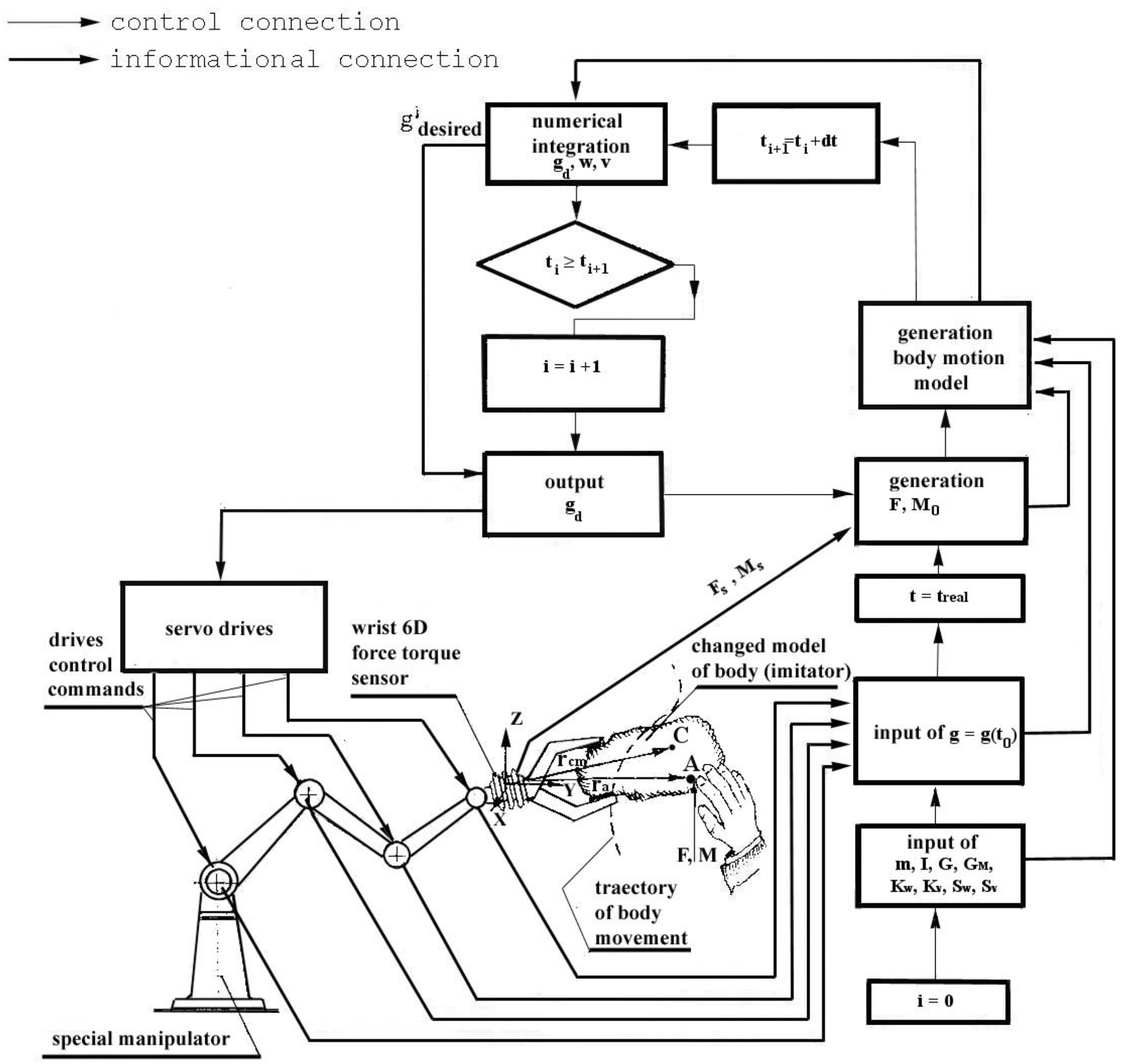


Fig.3.1 Block-diagram of an algorithm for simulating interaction of hand with virtual body's mock-up.

Equations of movement for a body having mass m and weight G in the form of differential Euler-Lagrange equations expressed in a vector-matrix form will be:

$$m\dot{v} + m\omega \times v = F - k_v v - \alpha(g)G, \tag{3.1}$$

$$\mathrm{I}\dot{\omega} + \omega \times I\omega = M_0 - k_\omega \omega, \tag{3.2}$$

$$\dot{g} = J(g)(\omega, v), \tag{3.3}$$

where

$(\omega,v) = (\omega_1,\omega_2,\omega_3,v_1,v_2,v_3)$ is 6D vector of angular $\omega=(\omega_1,\omega_2,\omega_3)$ and linear $v=(v_1,v_2,v_3)$ speeds of body in the body's frame of coordinates;

$g=(g_1,g_2,g_3,g_4,g_5,g_6)$ is 6D vector of joint coordinates of manipulator's handle;

$J(g)$ is 6×6 Jacobian matrix of manipulator which is a known function of manipulator's joint coordinates;

G – weight of body in the inertial system of coordinates;

$\alpha(g)$ - is 3×3 matrix of direction cosines determining rotation of body's coordinates system in the inertial one, it is also the known function of manipulator's joint coordinates;

k_v and k_ω are 3×3 diagonal matrices of body's coefficients of friction.

If there are non-golonomous constraints making a body to move over a surface, the body movement is determined, besides equations (3.1), (3.2) and (3.3), with equations of constraints having, generally, the following form:

$$(E-S_v)v = 0, \tag{3.4}$$

$$(E-S_\omega)\omega = 0.$$

Here S_v and S_ω are 3×3 selective diagonal matrices having over diagonals zeros and units, units corresponding to projections of speeds whose values are not constrained; it is obvious that equations (3.4) correspond to r scalar equations of a form ω_j=0 and v_j=0.

The system of equations (3.1), (3.2), (3.3) and (4) may be transformed to a form

$$S_v(m\dot{v} + mS_\omega\omega \times S_v v) = S_v(F - k_v v - \alpha(g)G),$$

$$S_\omega(I\dot{\varpi} + S_\omega\omega \times IS_\omega\omega) = S_\omega(M_0 - k_\omega\omega),$$

$$\dot{g} = J(g)(\omega,v).$$

These vector-matrix equations obviously correspond to 12 scalar equations, (6-r) in the first six being zero identities. The equations are completely defined if there are known body mass, inertia matrix I, center-of-mass positions in the body r_c and its mock-up r_{cm} in the body's coordinate sysytem and, also, mock-up's weight $G_м$. Thus, knowing these quantities and, also, initial values $\omega=\omega_0$, $v=v_0$, $g=g_0$, one can calculate ω and v and, also, g as functions of time using one of numerical methods for integration of differential equations.

The equations are even simpler if a body moves in weightlessness, then G=0 and $k_v=k_\omega=0$.

The obtained values of manipulator's joint coordinates may be used as desired values g_d for control systems of each joint coordinate's drive which track these values. In the result, all six coordinates will track these values with specified accuracy and, so, body's mock-up will with the calculated linear and angular speeds and, hence, will move so as a real body would do under action of the applied to it force and torque M.

The control device for simulating interaction of hands with virtual body operators to the following algorithm (Fig.3.1).

The values of mass m and inertia matrix I of virtual body are entered in the data processing unit of the device prior to operation. These values must be considerably greater than those of its mock-up actually moved by manipulator. Also the body's center-of-mass position r_c, weight G and the mock-up's weight G_M, friction matrices k_ω and k_v and, also, values of selective matrices S_ω and S_v are entered the unit. These data are needed for concretizing body's dynamic equations (3.1) and (3.2).

At application to the mock-up a force F and the torque M in the time moment t_0 the sensor measures torque M_s and force F_s. SW means of the data processing unit uses these data as input ones for computing mock-up's center-of-mass position r_c values of force F and torque M_0 applied to virtual body. It integrates the dynamic equations, viz. computes linear speeds of virtual body's center of gravity $v = (v_1, v_2, v_3)$ and its angular ones ω=(ω_1, ω_2, ω_3) for a moment t_1=t_0 +Δt (Δt being the step of computation) and, also, computes the value of the vector of joint coordinates g_d^1, for the manipulator. This value is the desired one for drives in time interval from t_1 to t_2=t_1 +Δt.

Let us note, that time consumed for computing the desired joint coordinates vector g_d^1, as every other value of g_d^i at subsequent time intervals, is always less than interval Δt between outputs of these calculated desired values. Therefore, the next output is at t=t_1. And upon that there begins the computation cycle, similar to that of previous interval i.e. the M_s and F_s measurement, calculation F_0, M_0 finding of the values v_1, v_2, v_3 and ω_1, ω_2, ω_3 for the time moment t_2=t_1 +Δt, calculation of desire coordinates g_i for servo drives on interval from t_2 to t_2 +Δt. In the moment t_2 the cycle of calculation repeats and etc. Obviously, the less the interval Δt, the less the delay in generation of the desired values for drives and hence, the error of physical imitation of body motion is less.

This inaccuracy is added to an error caused by limited dynamic capability of drives. The more the applied force and less virtual body's inertia and friction the more will be speed and, hence, the dynamic error and worse the simulation.

It is noteworthy, that the developed method and technical means for simulating the tactile-kinaesthetic interaction are purposed for the same functions as in known systems Cyber Force, Cyber Grasp, Cyber Glove produced by Virtual Technologies Company.

But unlike these systems, the proposed ones are significantly cheaper. Besides, the proposed means provide a more realistic simulation.

This is ensured by that the tactile-kinaesthetic perception and distinguishing shape and size of body are absolutely precise because the mock-up's size and form closely simulates those of virtual body and is perceived immediately with hands, unlike Cyber Glove and Cyber Grasp which do not provide feeling of form and size.

Besides that, using the force&torque sensor measuring resultant force and torque applied to the mock-up with hand and using a system for simulating the actual movement from action of applied force enable enough adequate feeling, even with a body in weightlessness.

System Cyber Force cannot provide such ample imitation.

Somewhat lesser universalism than that of Cyber Force caused by necessity of changing mock-ups with new virtual bodies, as experience shows, is not a grave miss because a list of bodies required for a specific task is limited and mock-up's price (e.g. made of foam plastics) is very low.

3.2 An approach to kineastaetic interaction with virtual manipulator

3.2.1 Base phases of kineastaetic interaction realization

In a case when virtual object immersed in real environment is a manipulator telecontrolled in master-slave mode, hand's interaction with this object has some peculiarities. Man controlling virtual manipulator uses a special mechanism having 6 degrees of freedom. Such mechanism may be the master arm kinematically similar to the manipulator. The manipulator's control system must provide manipulator's tracking position of the master arm, viz. a tool on the manipulator's end must copy master arm's position/orientation, precisely, its handle moved by hand. For realizing a more perfect copying control, so-called control with "reflection of efforts" [2], the manipulator's control system creates on the handle force and torque perceived by man which are equal or proportional to force and torque acted on the manipulator-end tool at interaction with environmental objects. Thus, while controlling virtual manipulator, man's hands do not touch objects (virtual and real), the manipulator comes in contact with, but perceive force and torque of their reaction by means of the master arm hand lie on. Therefore, unlike the previous case, hand's image is not to be generated what considerably simplifies the task of realizing the visual effect of immersion. But there arises a need to use the master handle moved by man.

This arm is to be kinematically similar to virtual manipulator and have joint drives controlled so that man would perceive at the handle the same force and torque as real manipulator would at the same situation.

Realizing this requires: first, determining if there is a contact of virtual manipulator with environmental objects and, so-called, type of the contact (impact) and its parameters characterizing the spot of impact; second, computing values of expected current forces and torques arising in virtual manipulator's interacting with environment and, also, its joint coordinates for this type of impact; third, generating such control signals for each drive as would create at the handle force and torque equal or proportional to the computed virtual ones.

The general algorithm for generating hand's interaction with virtual manipulator (Fig. 3.2) operates cyclically and comprises three main blocks.

The first block detecting fact of impact, its type and parameters utilizes two kinds of input data:

- entered prio to operation invariable data characterizing geometric models of environment and virtual manipulator;
- current data on virtual manipulator's target state, they are current values of master arm's joint coordinates obtained with the help of special position sensors mounted in joints of the master arm.

This block outputs:

- current values of virtual manipulator's joint coordinates which are equal to those of master arm when there is no impact;
- parameters describing contact's type and corresponding to it data determining impact's spot, these data are utilized by the second algorithmic block.

The second algorithmic block is one for computation of expected forces and torques of interaction.

Input data to this block are those of the first added with parameters characterizing contact's type and spot.

Output data are:

- Vectors of force and torque resulting in virtual manipulator's interaction with environment;
- Current values of joint coordinates at a fact of impact which are equal to a sum of two quantities, the first being current value of master arm's joint coordinates and the second – correcting value to the first caused by reaction of the obstacle.

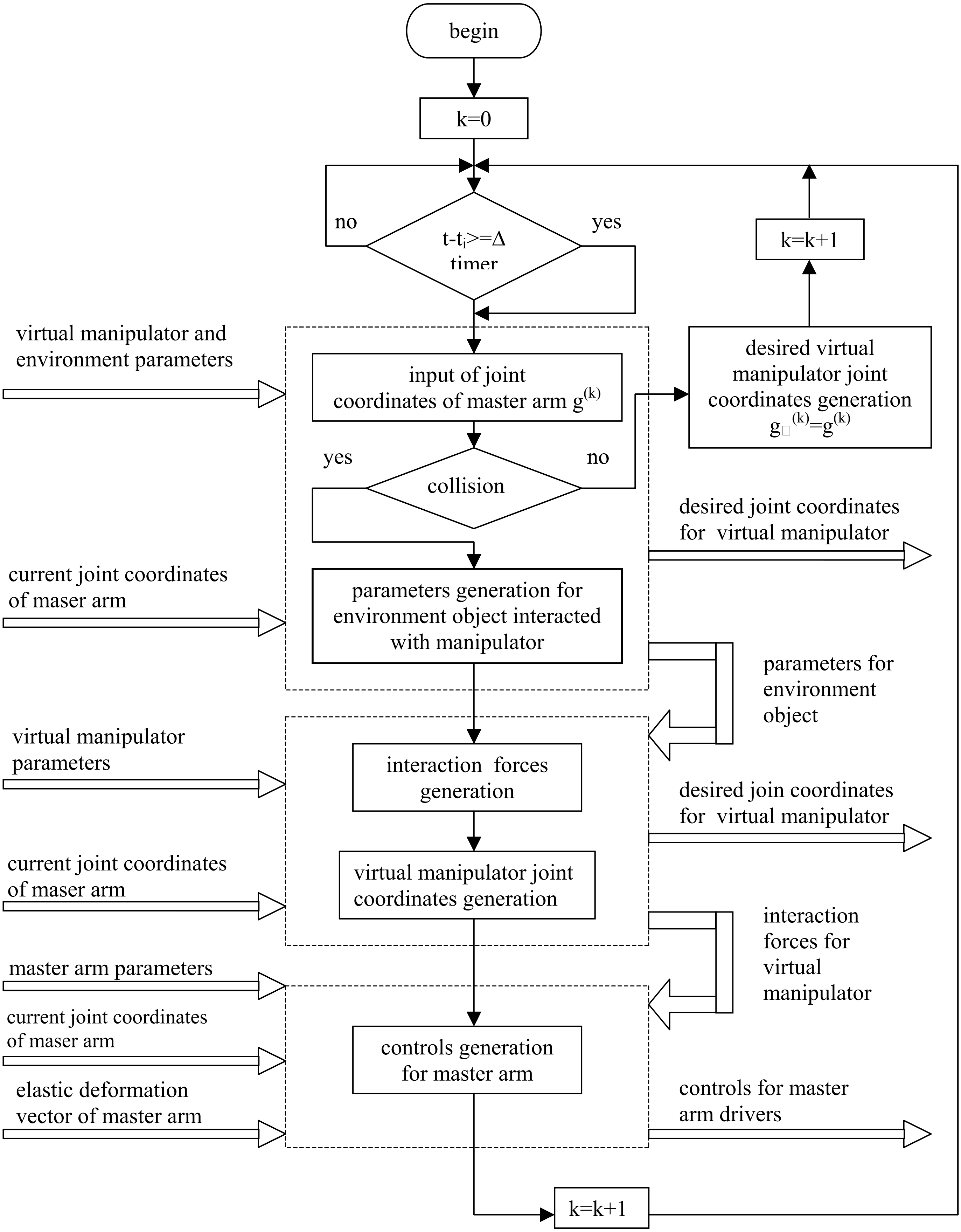


Fig.3.2 Block-diagram of general algorithms for generation of force interaction of human hands with virtual manipulators.

The third algorithmic block provides generation of control signals for the master arm's drives.

Data input to the block are:

- Entered prior to operation invariable data characterizing the arm as the object of control;
- Current data which are vectors of expected forces and torques of virtual manipulator's interaction with obstacles computed with block 2; master arm's joint coordinates obtained with joint's positional sensors and 6D vector of forces and torques of reaction measured with the force&torque sensor on the arm's handle or corresponding to it 6D vector at the arm's end, elastic arm's deformation measured with the help of special tensometric sensors placed on the arm in certain places.

Output data are those of control for each master arm's drives.

3.2.2 Detecting virtual manipulator's impacts with environment

Let us concretize the task of detecting impacts, admitting the following:

- Only the manipulator gripper's impacts will be considered and those of its links will not;
- The gripper will be modelled with a convex multihedron;
- Geometric model of environment will be a combination of spatial regions of two kinds, one of which is a conjunction of 3D hemispaces separating regions of possible and impossible gripper's positions and the second one is an aggregation of convex multihedrons one face of which belongs some outward hemispatial surface.

Thus, for providing the gripper's having no contacts with environment the following conditions are to be fulfilled (possible contacts are shown in Fig.3.3):

- No vertex of the convex multihedron modelling the gripper may be in regions occupied by obstacles;
- No vertex belonging to any of convex multihedrons modelling environmental objects may be in the region belonging to the convex multihedron modelling the gripper.

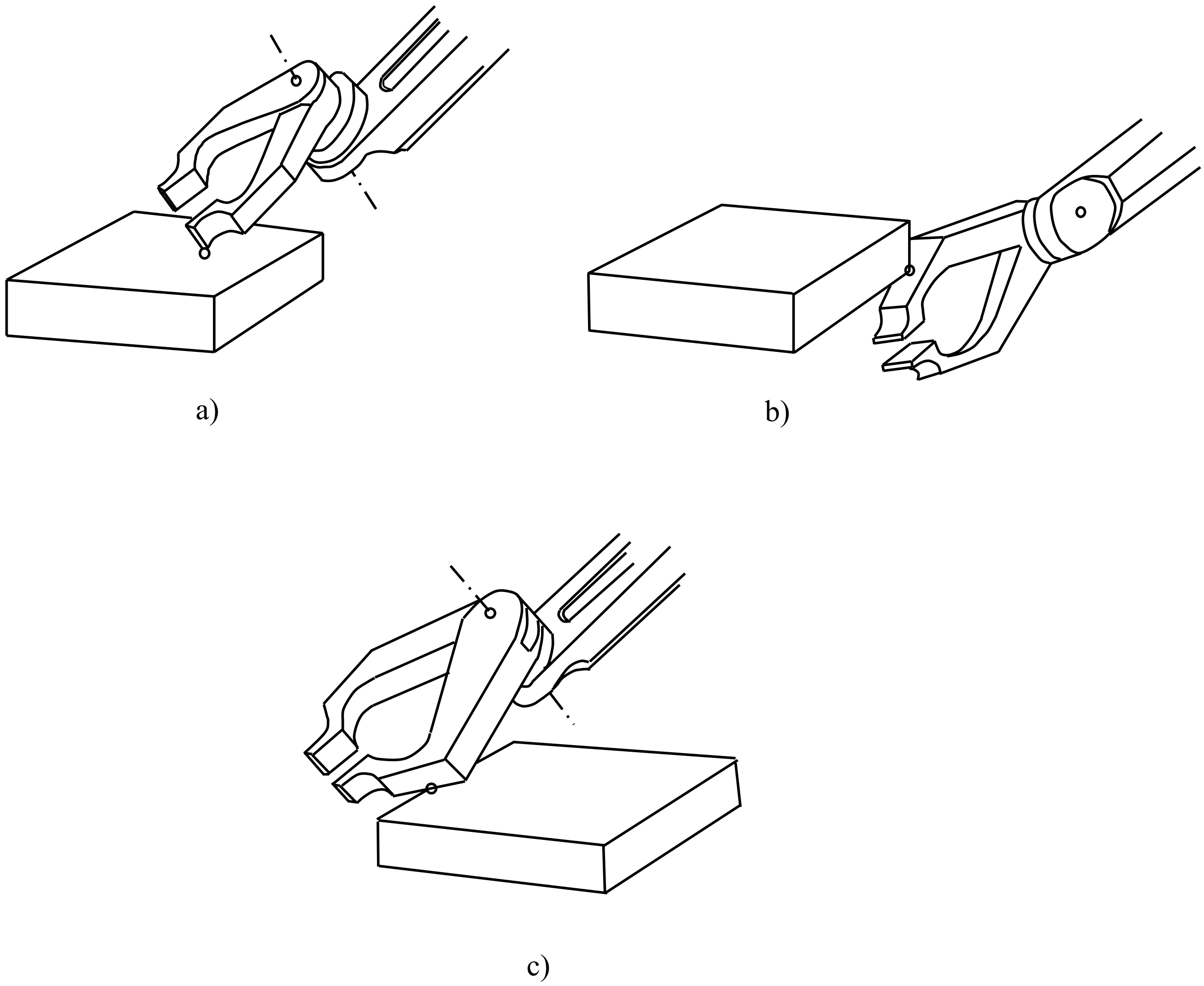


Fig.3.3 Possible variants of contact of manipulator's work tool with environmental objects.

The first condition is equivalent to fulfilling ($n \times r$) inequalities of the form:

$$q_j^{\mathrm{I}}(y_i - y_{sj}) > 0 \qquad j = 1,...,n;\ i = 1,...,r. \tag{3.5}$$

where y_i is a vector in the inertial coordinates system to the i^{th} vertex out of r vertices of the multihedron modelling the gripper; q_j^{I} and y_{sj} are, respectively, the outward unity normal to the j^{th} plane out of n planes bounding to the hemispaces modelling environment and the point on this plane; q_j^{I} and y_{sj} are given in the inertial coordinates system.

The second condition is equivalent to fulfilling ($k \times m$) inequalities of the form:

$$q_i^{\mathrm{II}}(x_j - x_{si}) > 0, \qquad i = 1,...,k;\ \ j = 1,...,m\ , \tag{3.6}$$

where x_j is a vector in the gripper's coordinate system to the j^{th} vertex out of m vertices of multihedrons modelling environmental objects;

q_i^{II} and x_{si} are, respectively, the outward unity normal to the i^{th} face out of k faces of the mutihedron modelling the gripper and the point on this face given in the gripper's coordinates system.

Thus, for establishing possible impacts one should test for verity inequalities (3.5) and (3.6).

For known geometric models of environment and the gripper values q_j^{I}=const, y_{sj}=const and q_i^{II}=cosnt, x_{si}=const are also known beforehand.

Therefore, establishing impacts requires computation of y_i position of each out of r vertices of the multihedron modelling the gripper in the inertial system of coordinates, x_j position of each out of m vertices of mutihedrons modelling environmental objects in the gripper's coordinates system and testing for verity inequalities (3.5) and (3.6). Evidently, positions x_j and y_i of the above vertices are functions of vector g of the manipulator's joint coordinates and the manipulator's kinematic model is to be used for determining these positions, one which sets relations between vector g and quantities determining position and orientation of each manipulator's link including the extreme one, i.e. the gripper. A detailed description of an universal kinematic model for a wide range of manipulation mechanisms with arbitrary number of degrees of freedom and the algorithm of its computer realization are given in work [24-25].

The gripper's orientation in the inertial system is determined with a matrix of direction cosines α and its position with vector y_0 to the origin of the gripper's coordinate system. Elements of matrix α and components of vector y_0 are functions of g. They are determined by means of the above kinematic model.

Values y_i and x_j are the following functions α and y_0:

$$y_i = \alpha\, y_i^{gr} + y_0, \tag{3.7a}$$

$$x_j = \alpha^{\mathrm{T}}(x_j^{v} - y_0), \tag{3.7b}$$

where y_i^{gr} and x_j^{v} are, respectively, known beforehand constants: vector to the i^{th} vertex of the multihedron modelling the gripper given in the gripper's coordinates system and vector to the j^{th} vertex of multihedrons modelling environmental objects given in the inertial coordinates system.

Thus, the algorithm of testing for fact of impact requires:

- Establishing a fact of impact for a current master arm's vector of joint coordinates and a type of impact;

– Determining values of unity normals q_j^{I} and q_i^{II} belonging to planes which came to the contact with the j^{th} vertex of the multihedron modelling the gripper, for the first type of impact, and with the i^{th} vertex of the multihedron modelling environmental object.

The testing is made cyclically every time interval Δt chosen for being enough for the testing procedure and yet possibly small for vector's g increment Δg over this interval not superpassing some small critical value.

3.2.3 Computing expected values of manipulator's forcible interaction with environment

The virtual manipulator's force interaction with environment may be described with a system of equations comprising the dynamic equations expressed in a vector-matrix form as:

$$\frac{d}{dt}\left(\frac{\partial \mathrm{T}}{\partial \dot{g}}\right) - \frac{\partial \mathrm{T}}{\partial g} + \frac{\partial \Pi}{\partial g} = u + \left(\frac{\partial R(g)}{\partial g}\right)^{\mathrm{T}} \lambda - V\dot{g} \tag{3.8}$$

and constrains equations:

$$R(g) = 0, \tag{3.9}$$

where g is the manipulators nd vector of joint coordinates,

λ is the md Lagrange's multiplier,

u is the nd control vector,

$\mathrm{T}(g, \dot{g})$ and $\Pi(g)$ are manipulator's kinetic and potentional energies,

$R(g)$ is md vector function,

$\left(\frac{\partial R}{\partial g}\right)^{\mathrm{T}} \lambda$ is the nd vector of generalized reactions,

$V\dot{g}$ is the nd vector of generalized dissipative forces of friction.

The constraints are golonomous ones they constraint a feasible motion of the gripper caused by contact with environmental objects.

The constraint equations are algebraic ones in g, vector of joint coordinates.

When a contact is of the first type and the unequalities (3.5) becomes equalities, the following constraint equation takes place

$$R(g) = R^{\mathrm{I}}(g) = (q_j^{\mathrm{I}})^{\mathrm{T}}(y_i(g)\text{-}y_{ni}) = 0, \tag{3.10}$$

where $y_i(g)$is determined with equation (3.7a) in which elements α and y_0 are known functions of g.

At the second type of contact the constraint equation have, evidently, a form:

$$R(g) = R^{\mathrm{II}}(g) = (q_i^{\mathrm{II}})^{\mathrm{T}}(x_j(g)-x_{ni}) = 0. \qquad (3.11)$$

For the manipulators quasistationary movement, when all dynamic and dissipative forces entering in (3.8) are insignificantly small, the equations will take a form:

$$\Psi(z) = \begin{cases} k_y(g_d - g) + \left(\dfrac{\partial R}{\partial g}\right)^{\mathrm{T}} \lambda = 0 \\ R(g) = 0 \end{cases}. \qquad (3.12)$$

They are obtained taking in account that dependent from $\dot{g}$ and $\ddot{g}$ components of vector U are by a master-slave control insignificantly small and the control law takes a form:

$$U = k_y(g_d - g) + \frac{\partial \Pi}{\partial g}, \qquad (3.13)$$

where k_y is the gain matrix,

g_d is nd vector of target virtual manipulator's joint coordinates tasked with the master arm,

$\frac{\partial \Pi}{\partial g}$ is a component compensating an effect of potential forces begot by weight of manipulator's links and a weight borne by manipulator's[1]. System (3.13) is a system of $(m+n)$ algebraic nonlinear equations in an unknown vector Z = $(q_1, q_2,..., q_n, \lambda_1, \lambda_2,..., \lambda_m) = (q, \lambda)$.

The unknown values of force and torque vectors appearing in virtual manipulator's impact environmental object is a function of component λ of this vector which will be determined below.

It is expedient to use the Newton's method for finding Z, which is based on linearization of system (3.13) in the region of a known current value of vector $Z=Z^{\mathrm{lin}}=(q^{\mathrm{lin}}, \lambda^{\mathrm{lin}})$, and determining the unknown Z as the solution of this linearized system by a method of successive approximations.

The linearized system of equations (3.13) in point $Z=Z^{\mathrm{lin}}$ has a form:

$$\psi^{\mathrm{lin}}(Z) = \psi(Z^{\mathrm{lin}}) + \frac{\partial \Psi(Z^{lin})}{\partial Z} \Delta Z = 0, \qquad (3.14)$$

[1] Not limiting the general character of interpretation we may put that potential forces caused by elasticity of manipulator's construction are absent.

where

$\Delta Z = (\Delta q_1, \Delta q_2,..., \Delta q_n, \Delta\lambda_1, \Delta\lambda_2,..., \Delta\lambda_m)$,

$$\frac{\partial\Psi(Z^{lin})}{\partial Z} = \left\| \begin{array}{c|c} -k_y & (C_{lin})^{\mathrm{T}} \\ \hline C_{lin} & 0 \end{array} \right\|,$$

$$C_{lin} = \left(\frac{\partial R(g)}{\partial g} \right)^{\mathrm{T}} \Bigg|_{g=g_n^{lin}},$$

$$\psi(Z^{lin}) = \left\| \begin{array}{c} -k_y(g_d - g^{lin}) + C_{lin}\lambda^{lin} \\ R(g^{lin}) \end{array} \right\|. \tag{3.15}$$

For the first type of contact we get taking in account (3.10) and (3.7a):

$$C_{lin} = C_{\mathrm{I}} = \frac{\partial R^{\mathrm{I}}(g)}{\partial g} = (q_j^{\mathrm{I}})^{\mathrm{T}}\mathrm{J} + (q_j^{\mathrm{I}})^{\mathrm{T}} \frac{\partial(\alpha y_i^{gr})}{\partial g}, \tag{3.16}$$

and for the second type, taking in account (3.11) and (3.7b), we have

$$C_{lin} = C_{\mathrm{II}} = \frac{\partial R^{\mathrm{II}}(g)}{\partial g} \text{ - } (q_i^{\mathrm{II}})^{\mathrm{T}} \bar{J} + (q_i^{\mathrm{II}})^{\mathrm{T}} \frac{\partial(\alpha^{\mathrm{T}} x_j^v)}{\partial q}, \tag{3.17}$$

where $J = \dfrac{\partial y_0}{\partial g}$ and $\bar{J} = \dfrac{\partial(\alpha^{\mathrm{T}} y_0)}{\partial g}$ are Jacobean matrices of virtual manipulator.

As Z^{lin}, corresponding to manipulator’s transition from free movement to the “constrained”, i.e. when the gripper comes in contact with environmental object and the equations $R(q)=0$ for constraints are to be fulfilled, we shall take a point $Z^{lin} = Z^0 = (g^0, \lambda^0)$, where g^0 corresponds to the gripper’s touching environmental object not exercising any force. Evidently, value λ corresponding to g^0 will be 0, i.e. $\lambda=\lambda^0=0$ and $Z^0=(g^0,0)$.

Respecting the determination of quantity g^0 it should be noted the following.

A current value of vector g at “free” movement of virtual manipulator precisely copies that of the master arm’s vector of joint coordinates which are measured with position sensors and are inputted to the algorithm’s block detecting impacts. But sampling values g_d from the buffer is made in discrete moments of time separated with intervals Δt. During each interval value of g_d (and, consequently, q) will change at $\Delta g_d = \Delta g$. Therefore, values of g entered in the block cannot, in principle, satisfy the constraint equations $R(q)=0$ and a fact of virtual robot's contact with environment is established by changing of sign $R(q)$.

The method for precise determination of value g^0 at the first gripper's contact with environment is based on a probable assumption that on a small interval between checks vector g changes linearly. Then

$$g = g^{(-1)} + \mu(g_d^{(0)} - g^{(-1)}), \qquad 0< \mu < 1, \tag{3.18}$$

where $g_d^{(0)}$ is a value of vector g_d for initial step in detecting impacts, i.e. for which $R(g)$ changed its sign;

$g^{(-1)}$ - value of g on the preceding step.

Evidently, the unknown will be

$$g^0 = g^{(-1)} + \mu_0(g_d^{(0)} -g^{(-1)}),$$

where μ_0 satisfies any equation in system $R(q)=0$ in which argument g is substituted for its value (3.17).

Solution of system (3.14)

$$\Delta Z = \left\| \frac{\partial \Psi(Z^{lin})}{\partial Z} \right\|^{-1} \Psi(Z^{lin}) \tag{3.19}$$

is available if rank $\frac{\partial \Psi(Z^{lin})}{\partial Z}$ is equal to the rank of characteristic matrix

$$\left\| \begin{array}{c:c} \frac{\partial \Psi(Z^{lin})}{\partial Z} & \Psi(Z^{lin}) \end{array} \right\|.$$

Utilizing the Frobenius' formula [26] and assuming k_y being scalar and

$$\left\| \frac{\partial \Psi(Z^{lin})}{\partial Z} \right\|^{-1} = \left\| \begin{array}{c:c} -k_y^{-1} + k_y^{-1} C_{lin} (C_{lin} C_{lin}^{\mathrm{T}})^{-1} C & C_{lin}{}^{\mathrm{T}} (C_{lin} C_{lin}^{\mathrm{T}})^{-1} \\ \hdashline (C_{lin} C_{lin}^{\mathrm{T}})^{-1} C_{lin} & k_y (C_{lin} C_{lin}^{\mathrm{T}})^{-1} \end{array} \right\|. \tag{3.20}$$

At $Z^{lin} = Z^0 = (g^0, \ \lambda^0)$ on the first step of computation after gripper's impact with environment $\lambda^0=0$, $R(g^0) = 0$ and, consequently according to (3.15)

$$\Psi(Z^{\mathrm{lin}}) = \Psi(Z^0) = \left\| \begin{array}{c} -k_y (g_d^0 - g^0) \\ \hdashline 0 \end{array} \right\|. \tag{3.21}$$

Then from expression (3.19) for ΔZ taking in account (3.20), (3.21) we get

$\Delta Z = Z' = (\Delta g^1, \Delta\lambda^1)$,

where

$\Delta\lambda^1 = -\, k_y(C_0C_0^T)^{-1}\, C_0\Delta\, g_d^0$

$\Delta g^1 = \Delta\, g_d^0\; = C_0^T(C_0C_0^T)^{-1}\, C_0\Delta\, g_d^0$;

here the following equations are used:

$\Delta\, g_d^0 = g_d^0 - g^0, \quad C_0 = C_{lin} \quad$ при $\quad g^{lin} = g^0$.

On completing one iteration for determining λ, g corresponding to values $g_d = g_d^0$, $g^{\text{lin}} = g^0$, $C_{lin} = C_0$, the unknowns will be

$$\lambda^1 = 0 + \Delta\lambda' = -\, k_y\, (C_0C_0^T)^{-1}\, C_0\Delta\, g_d^0\,, \tag{3.22}$$

$$g^1 = g^0 + \Delta g' = g_d^0 - C_0^T(C_0C_0^T)^{-1}C_0\Delta\, g_d^0\,. \tag{3.23}$$

The next block's sampling of g_d for determination of the next values will be after a time Δt, therefore a value of g_d used for computing λ and g will change and becomes g_d' as the master arm is moved by man. If one uses values $g^{lin}=g^1$ and $\lambda^{lin}=0$ as a point for linearizing equations (3.12) in which $g_d = g_d'$ then unknown values $g = g^2$ and $\lambda = \lambda^2$ determined from the linearized equations (3.14) on the first iteration will satisfy the formulas analogous to (3.22) and (3.23) in which $\Delta\, g_d^0$ is substituted for $\Delta g_d' = g_d' - g'$, and matrix C_0 for $C_1 = \left.\left(\dfrac{\partial R}{\partial g}\right)^T\right|_{g=g'}$.

Evidently, in any k^{th} moment the unknown values $g=g^k$ and $\lambda=\lambda^k$ may be found with the same formulas (3.22), (3.23) substituting C_0 for C_{k-1} and $\Delta\, g_d^0$ for Δg_d^{k-1}, which provide practically tolerable accuracy of funding the unknown values λ and g

$$\lambda^k = -k_y(C_{k-1}\, C_{k-1}^T)^{-1}C_{k-1}\Delta g_d^{k-1}, \tag{3.24}$$

$$g^k = g_d^{k-1} - C_{k-1}^T(C_{k-1}\, C_{k-1}^T)^{-1}C_{k-1}\Delta g_d^{k-1}. \tag{3.25}$$

The unknown expected value of vector F resulting from virtual manipulator's interacting with environment is determined with the obtained value of λ. The according expression for F results from a condition of equality of works: that of generalized forces' vector $\left(\dfrac{\partial R}{\partial g}\right)^T \lambda$ over path δg and that of the unknown vector of forces F applied in the point of gripper's contact with environmental object over virtual path δl gone by this point of contact

$$\left(\frac{\partial R}{\partial g}\right)^{\mathrm{T}} \lambda \cdot \delta q = F \cdot \delta l. \tag{3.26}$$

Taking in account respective expressions for δl for contacts of the first and second types: $\delta l^{\mathrm{I}} = \dfrac{\partial y_i}{\partial q}\delta q$ and $\delta l^{\mathrm{II}} = \dfrac{\partial x_j}{\partial q}\delta q$ and substituting them in (3.26) we shall find the following expressions for vector of force F in the point of contact:

$F^{\mathrm{I}} = q_j^{\mathrm{I}}\lambda$ for the first type,

$F^{\mathrm{II}} = q_i^{\mathrm{II}}\lambda$ for the second type.

As the matter of course, F^{I} is given in the inertial coordinates systemf and F^{II} - in that of gripper.

In the gripper's system it is just for both types:

$$\overline{F}^{\mathrm{I}} = \alpha^{\mathrm{T}} F^{\mathrm{I}} = -\, k_{\mathrm{y}} \alpha^{\mathrm{T}} q_j^{\mathrm{I}} (\mathrm{C_I C_I^T})^{-1}\, \mathrm{C_I} \Delta g_d, \tag{3.27}$$

$$\overline{F}^{\mathrm{II}} = F^{\mathrm{II}} = k_{\mathrm{y}} q_j^{\mathrm{II}} (\mathrm{C_{II} C_{II}^T})^{-1}\, \mathrm{C_{II}} \Delta g_d\,, \tag{3.28}$$

where $\mathrm{C_I}$ and $\mathrm{C_{II}}$ are computed respectively with formulas (3.16) and (3.17) in which terms J, $\bar{J}$, $\dfrac{\partial(\alpha y_i^{gr})}{\partial g}$, $\dfrac{\partial(\alpha^{\mathrm{T}} x_j^{v})}{\partial g}$ are functions of g. They are computed for $g=g^{k-1}$ and $\Delta g_d=\Delta g_d^{k-1}$.

3.2.4 Generating control signals for the master arm's drives

A special control system is needed for generating control signals for joint drives of master arm providing at its handle a reactive force equal to the computed one. In fact, it is a typical system of forces&torque control, or a compliant motion control system with master arm's being the object of control. This object has constraints because hand holding the handle constraints its all feasible movements. The desired force reaction must be the computed expected vector of force applied by virtual manipulator interacting with environment transferred to application point on the handle. And the controlled quantity, used also as the feedback signal, is to be the vector of forces and torques produced in the application point by master arm's joint drives and measured with a special force&torque sensor on the handle. To-day methods for compliant motion control are given in detail in [16], including that of virtual robots [3] they are described in detailed also in Interim Report # 4 of Task 6. Therefore, these methods are not considered in this section. We only note, that the expected forces computed with formulas (3.27) and (3.28) for generating the appropriate hand's force reaction are to be related to a certain point of manipulator's gripper. E.g., this point may be the system's origin. Then, the handle's application point will be also the origin of the handle's system. Bringing force interaction to this equivalent point will, surely, adds to the computed forces torques.

For the first type it will be, evidently,

$$M^{\mathrm{I}} = F^{\mathrm{I}} \times (\alpha y_i^{gr}); \tag{3.29}$$

and for the second type

$$M^{\mathrm{II}} = F^{\mathrm{II}} \times [\alpha^{\mathrm{T}}(\mathrm{x}_j^{v} - \mathrm{y}_0)] \; . \tag{3.30}$$

Chapter 4 A prototype of experimental facility for verification of the technology for virtual object's immersion in real environment

4.1 Hardware

For experimental testing of the proposed technology a prototype was created of the hard&software facility whose structure is shown in Fig.2.1. It corresponds to a variant of the technology which deals with a virtual manipulator as the object. Its photo is shown in Fig.4.1.

Fig.4.1 The prototype of the complex for realization of virtual body immersion.

The functional units of this complex, briefly described in section 2.1, were realized on the base of the following hardware means.

<u>Unit 1.</u> The video cameras SANYO VCC-43-112P were used in the mobile stereo pair for imaging work scene and an anthropomorphous electromechanical manipulator of "PUMA" type was used as the robot-like device which bears them. Its gripper was substituted for a special platform on which the cameras were mounted on.

Basic manipulator's characteristics:

- Number of degrees of freedom - 6
- Maximal carrying capacity - 25 kg
- Repeatability of the manipulator's end positioning - ±0.1 mm
- Speed with maximal charge:
  - Over arbitrary trajectory - 0.5 m/s
  - Over linear trajectory - 1.0 m/s
- Operating zone envelope – sphere with radius 0.92 m

<u>Unit 2.</u> A conventional control system "Sfera-36", purposed for control of robot "Puma", is used for orientating/positioning the platform with the cameras imaging work scene (Fig.4.2).

Fig. 4.2 Robot drives control unit SPHERA-36

The system has two hierarchically connected control levels and six servodrives for each degree of freedom.

The upper level is realized on the base of the central processor executing the following functions:

– entering via RS-232 interface current coordinates of head’s position/orientation (generated in Unit 3) to Units 7 and 8 (graphic stations)

– computing six joint coordinates of the manipulator bearing the stereo pair utilizing the entered six coordinates of head’s position/orientation.

<u>Unit 3.</u> The optical TV system for acquisition of head’s position/orientation uses for it three active IR reference marks placed on the operator’s headband (Fig. 4.3). The marks are imaged with a video camera KPC-700 and processed with PC (Pentium III 733 MHz, Chipset Intel’s 815 Bx, Diskmemory 30 GB, RAM 256 MB, Videoadapter Asus - V7100 AGP 32Mb , Monitor 15 “ Sony E100).

Output PC data are current coordinates of head’s position/orientation serving as input ones for controlling position/orientation of the stereo pair imaging work scene (Unit 2).

Fig.4.3 3D viewers and colored marks for active optical HTS.

Unit 6. An optical microdisplay module DH-4400 mounted on operator's helmet (Fig.4.4) is used for producing alternatively to left and right eye stereo images of virtual manipulator "immersed" in the real environment. It has two microdisplays SVGA with resolution 800×600, diagonal FOV size 31,2°, eye base adjustable, supports any PC having a VGA port. Images may be displayed both on the microdisplays and graphic station monitors.

Fig.4.4. Twin displays of a type stereo-displays DH-4400 VPDV.

Technical Specification: Cy-visor DH-4400

Core Specifications

- VGA/SVGA/NTSC/PAL/S-VHS input capability
- SVGA high resolution display output
- 0.49-inch, 1.44 million dots reflective MicroDisplay
- Simulates dynamic 44-inch screen as viewed at 2m
- 25mm eye relief
- ± 31.2 ° degree diagonal FOV
- 13mm exit pupil

Input Signal

	Display Mode	Resolution (pixel)	Horizontal Freq.(KHz)	Vertical Freq.(Hz)
PC	VGA graphic	640 × 480	31.5	60
	VGA text	640 × 480	31.5	70
	VESA SVGA(60HZ)	800 × 600 60	37.9	60
	VESA SVGA(72HZ)	800 × 600	48.1	72
	VESA SVGA(75HZ)	800 × 600	46.9	75
Video	NTSC	All Formats		
	PAL	All Formats		
	SVHS	All Formats		

Output Signal

	Resolution (pixel)	Vertical Freq.(Hz)
Output	800 × 600	70

Specification

MODEL	**DH_4400**
Signal System	VGA/SVGA/NTSC/PAL/S-VHS
LSD	0.49-inch 1 .44 million dot microdisplay
Dot Number/panel	800(H) × 600(V) × RGB
Virtual Image Size	44-inch at 2m
Viewing Angle	± 31.2° degree diagonal
Video Input	Y/C, Composite
PC Input	RGB
Control Function	BRIGHTNESS
	CONTRAST
	COLOR
	VOLUME
Connectors	PC input D-sub 15PIN
	A/V input : Stereo Mini Jack
Power consumption	Approx. 3.3W(PC mode)
	Approx. 3W(AV mode)
Operating Temperature	10 ~ 50 °C
Storage Temperature	0 ~ 60 °C
Dimensions	Headset : 175mm(W)×225(D)×100(H)
	Controller : 58mm (W)×140(D)×45(H)
Weight	Headset : 160g
	Controller : 180g
Supplied Accessories	AC power adaptor
	AV. PC cables
Power	DC 9V

General

The DH-4400 display device is a complete optical microdisplay module that is available to HMD developers who require a turnkey solution for developing compact display headsets. It provides a 31.2-degree diagonal field of view (FOV) with a micro display that provides SVGA resolution (800 × 600).

The DH - 4400 provides viewing comfort with adjustable interpupillary distance (IPD) and comfortable eye relief. It can be directly connected to any computer with a VGA port.

Device Description

- The "XS-FT1" is a single chip solution for interfacing to the MicroDisplay.
- It performs several functions including frame rate conversion, format conversion, de-interlacing and time-sequential conversion.
- Algorithm of format conversion is FreeScale based on user freely adjustable interpolation function.
- It provides good quality scaled imagery better than fixed-interpolation functions
- It can be used for various applications such as monocular and binocular headsets for mobile computing, entertainment, and industrial wearable computing.

Features

- Input frame rate: Any frame rate can be converted to output frame rate 70Hz
- Output signal: Standard VESA SVGA (V-freq. 70Hz) digital RGB & SYNC for LCD panel
- Programmable input and output timing parameter
- User adjustable interpolation function (FreeScale algorithm)
- De-interlacing algorithm based on Spatial Vector Correlation
- Convert spatial signal to time sequential format
- Operation speed up to 100Mhz
- RGB or YUV input selectable.

Format conversion

- Fully programmable input timing parameters
- Any input acceptable lower than VESA SVGA 75Hz
- User adjustable interpolation function seed data (FreeScale algorithm)
- Can adjust seed data for high image quality
- Parallel and pipelined processing for high performance
- Any size image convertible to SVGA 800 × 600 (Horizontal, Vertical)

Frame rate conversion

- Any frame rate can be converted to 70Hz (Down or Up)
- Frame rate conversion based on Keep & Drop algorithm
- Input window cropping (adjustable with parameters)
- 24-bit RGB or 16-bit YUV input selectable

De-interlacing

- Spatial Vector correlation algorithm
- NTSC / PAL interlaced TV signal converted to process image
- Real-time processing with pipelining

Time-sequential signal conversion

- Converts spatial signal to time sequential format
- High speed processing over 100Mhz
- Single chip solution for the MicroDisplay

Unit 7 and Unit 8. Two PCs are used as graphic stations generating work scene images augmented with images of virtual object (for left and right eye) (Fig.4.5). Their characteristics are: Pentium III 800 МГц, Chipset Intel's 440 Bx, Videoadapter ASUS-V6800, DDR AGP RAM32Mb, Monitor 19" Sony E400, Videoinput – AverMedia EZCapture.

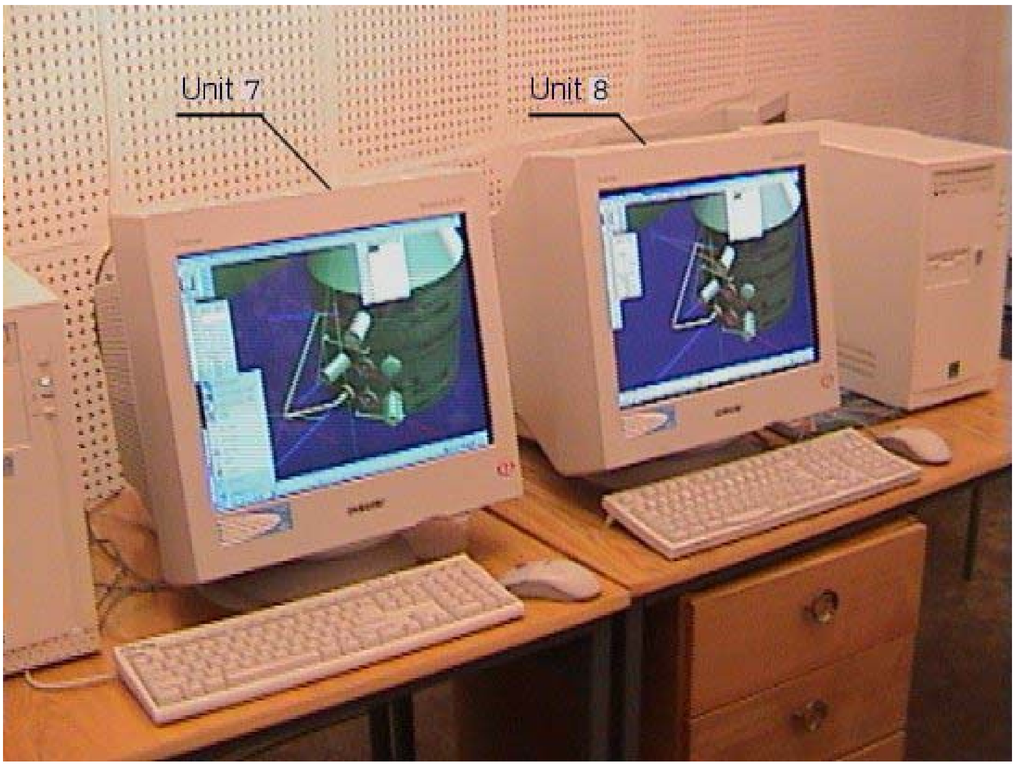


Fig.4.5. Photograph of Graphic Stations 1 and 2 (Units 7 and 8).

Unit 9. A manipulator of "PUMA" type is used as the master arm simulating virtual manipulator's force interaction with environmental objects, one kinematically similar to the virtual manipulator. Its gripper is substituted for a handle (Fig.4.6) with whose help the operator controls position/orientation of virtual manipulator's work tool.

Besides that, there is a special wrist force& torque sensor (Fig.4.7) measuring the effort with which man moves the arm, one the virtual manipulator exercises when it meets with environmental objects.

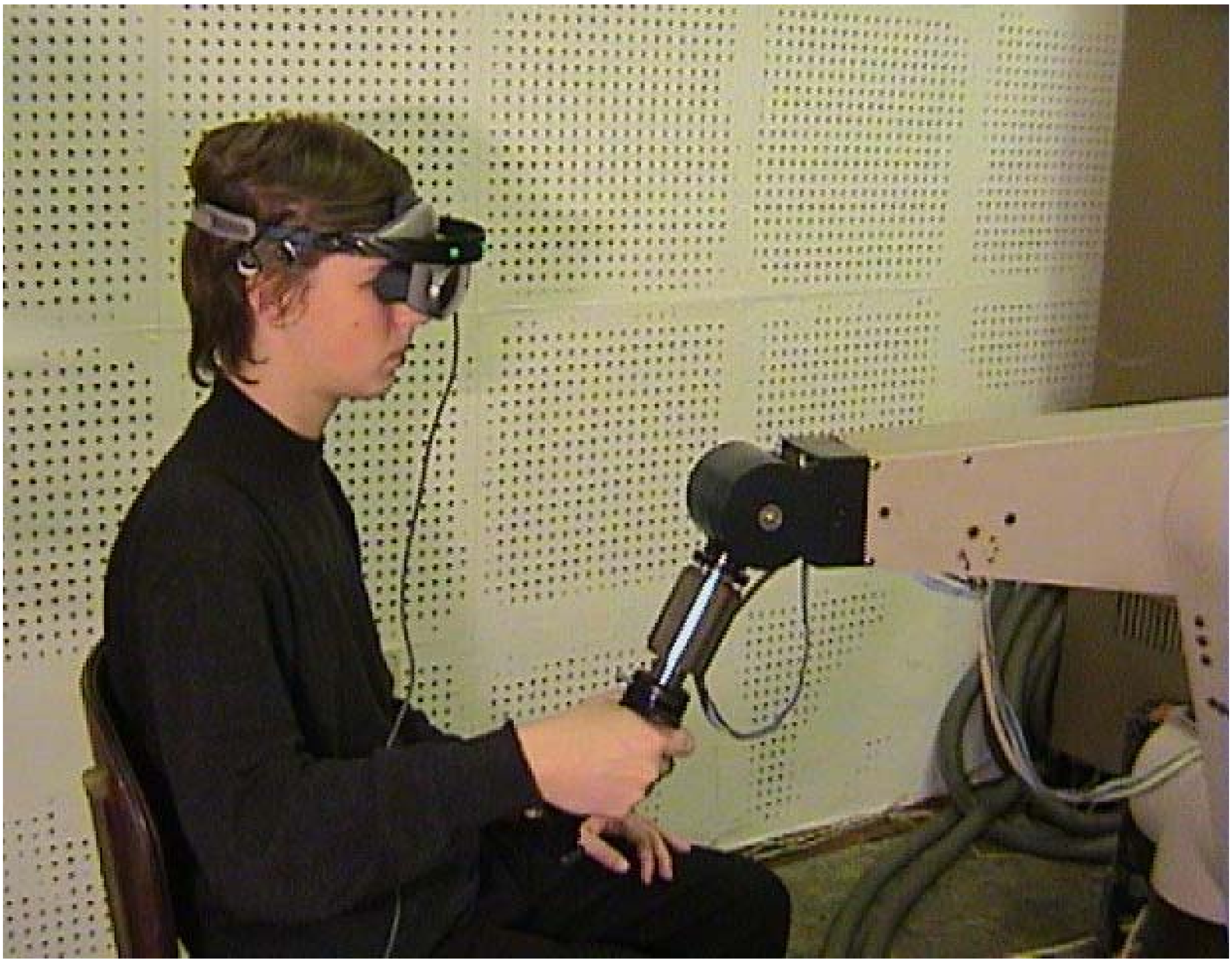


Fig.4.6 A special manipulator which is master arm for the virtual manipulator.

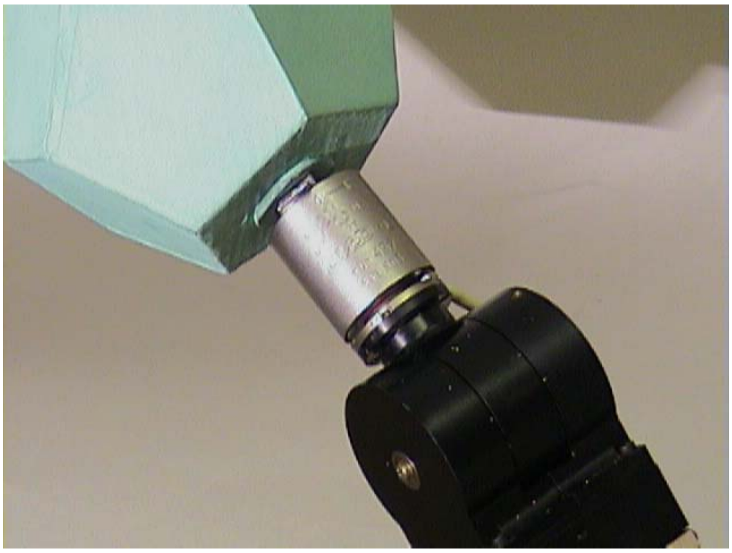


Fig.4.7. 6D wrist sensor.

Its technical characteristics are given above for it is the same one as used in Unit 1. The wrist sensor's technical characteristics are given in Table 1.

Technical characteristics of the wrist force sensor *Table 1*

Specifications of force/torque sensor						
Parameters and units of measurements	Specification values					
	X	Y	Z	M_x	M_y	M_z
Nominal load, N $N{\times}M$	350	500	500	5	10	10
Ultimate load, %	10	10	10	20	15	15
Sensitivity, mV/N $mV/N{\times}M$	0,0148	0,0111	0,0113	0,645	0,444	0,455
Total error, %	0,5	0,5	0,5	0,7	0,5	0,6
Operating temperature range,°*C*	-60 ÷ +70					
Gross dimensions, *mm*	diameter – 53; height - 62					
Weight, *g*	500					

Processing system specifications	
Power voltage, *V*	15
Number of motion channels	6 (8)
Number of binary digits of data bus	10
Number of binary digits of address bus	3
Conversion time, *Ms*	0.5

The master arm's drives are controlled with the above system "Sfera-36" but having functions different to those of Unit 1.

The upper level here generates:

- virtual manipulator's vector of joint coordinates whose values enter Graphic Station 1 to be used for generating virtual manipulator's image;
- expected force-and-torque vector of virtual manipulator's interaction with environmental objects;
- control signals for master arm's drives as functions of the expected force-and-torque vector, the feedback signals being force and torque measured with the wrist sensor;
- values of joint coordinates of each of six servo drives executing these values.

The lower level realizes the physical control of each of six master arm's joint drives utilizing as master signals those of the upper level and as the feedback – current values of joint coordinates.

The above hardware means supporting the complex' software have the following data exchange links.

<u>Unit 1.</u> The mobile video cameras imaging the work scene are connected with graphic stations 1 and 2 (units 7 and 8) via one way channels transmitting work scene's video images received with video capture cards AverMedia EZCapture.

<u>Unit 2.</u> The system controlling position/orientation of the mobile video cameras realized on the base of conventional control system "Sfera-36" is connected via a series interface R-232 with graphic stations 1 and 2 (units 7 and 8) for receiving coordinates of the cameras' position/orientation and, also, via a similar interface with Unit 3 for obtaining data on position/orientation of head.

<u>Unit 3.</u> The system tracking head's position/orientation and, specifically, data processing device based on PC Pentium III, is connected via one way channel with video cameras imaging head's reference marks, receiving card AverMedia EZCapture. Besides that, the system is connected with Unit 2 via serial interface R-232 for transmitting data on head's position/orientation.

<u>Unit 6.</u> It is realized on the base of optical microdisplay module D4-4400 and connected with graphic stations 1 and 2 (units 7 and 8) via one way channel for receiving work scene digital images augmented with that of virtual manipulator.

<u>Unit 7 and 8.</u> They are the graphic stations connected with the following units:

- Unit 1. One way channels for receiving work scene's video images from the mobile cameras;
- Unit 2 (control system). Serial interfaces R-232 for receiving coordinates of the mobile cameras' position/orientation;
- Unit 6. One way channels for transmitting work scene's digital images augmented with those of virtual manipulator to the optical micro display unit;
- Unit 9 (system for forcible interaction). Serial interface R-232 for receiving current values of virtual manipulator's joint coordinates and parameters characterizing virtual manipulator's impacts with environmental objects from the unit's control system.

Unit 9. Data exchange links are only those with units 7 and 8 given above.

4.2 Algorithms and software of the developed prototype of experimental complex for augmentation of virtual object to real environment

Basic parts of the developed to-day prototype of SW, installed at graphic stations 1 and 2 (Unit 7 and 8) and HTS computer for processing head position/orientation data of Unit 3, at Unit 2 controlling position/orientation of the mobile TV cameras, at master arm control system of Unit 9, and, also, information-and-control communication channels uniting these SW parts, their links with standard and non-standard devices for inputting/outputting data are shown in Fig.2.1. A description is given below of all listed SW parts, for exception of SW for HTS computer whose description is given in Report # 6 on Task 5.

4.2.1 The algorithm and SW prototypes for the master arm control system of Unit 9 realizing force interaction with man

Basic functions of SW prototype realizing the master arm control are cyclic computations of the following values:

1. Vector g_d of master arm's generalized coordinates used for graphic representation of virtual robot while it moves in the obstacle-free space (6 words to graphic station).
2. Vector Δg_c correcting master arm's generalized coordinates which being summed with the earlier transmitted vector g_d is used for graphic representation of virtual robot in its contact with an obstacle (3 words to graphic station).

3. Expected vectors of forces F_d^{I} and torques M_d^{I} of the virtual robot interaction with obstacles. The realized version of the subsystem for the virtual manipulator's interacting with man by means of the master handle is designed not for three, as was earlier planned (see Report # 3, Task 6), but for one possible type of work tool's collision with environmental objects, precisely, for collision of vertex of polyhedron approximating the tool with a surface (the first type of interaction according to the classification given in part 1.2 Report # 3 on Task 6). Moreover, this vertex is one and is the tool's tip lying on the axis of symmetry.
4. Control law providing a compliant master arm's motion utilizes difference between the calculated vector $G_d = (F_d, M_d)$ and current vector of operator's interaction with the master arm measured with force-torque sensor for generation of joint coordinates of master arm's vector g which must be executed by the lower level of the master arm's control system.

Input data for computing the abovementioned values are:

- indication (flag) of contact with an obstacle (1 Byte coming from graphic station);
- coordinates of point of virtual manipulator's contact with an obstacle in its base system of coordinates (3 words coming from graphic station);
- direction cosines of the normal q_j^{I} to j-th surface in the point of contact (3 words coming from graphic station);
- current values of the force F and torque M vectors of operator's interaction with the master arm (6 words coming from the force-and-torque sensor);
- vectors of generalized joint coordinates for the virtual robot and the master arm computed on the previous step (without contact are identical).

Fig.4.8 shows a structural block-diagram of the proposed cyclic algorithm. It should be noted at once, that situations, when algorithmic modules co-operating in data exchange do not use for calculation full data, have no influence on the protocol of exchange. Owing to that, the graphic station generates graphic images on every step (cycle) of algorithm, what simplifies work both for this module and software of the graphic station.

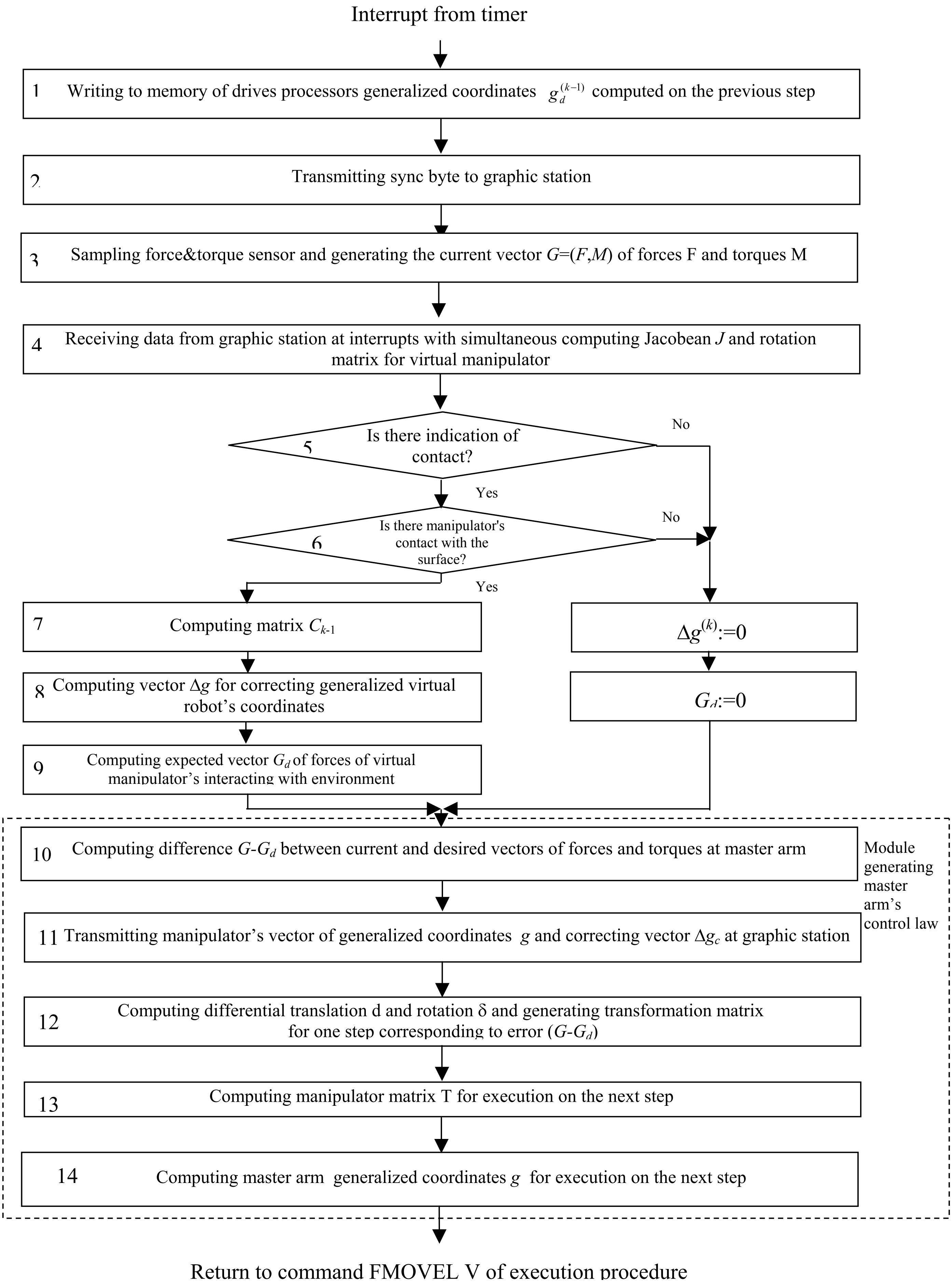


Fig. 4.8. Block-diagram of the algorithm for generating virtual robot's interaction with environment

In the following text a label "(block n)" points to the *n*-th block in the diagram.

Every operation step of the algorithm begins with writing to the memory of lower-level processors in control system SPHERA-36 values of generalized coordinates of the master arm computed on the preceding step (k-1) (block 1 of the algorithm). This is implemented by means of the module for data exchange between the upper and lower computing levels realizing direct addressing to the memory of the abovementioned processors. These computed values of generalized coordinates are components of the desired vector of the master arm's generalized coordinates $g_d^{(k-1)}$ for the (k-1)th step to be executed by drives of joint coordinates of the master arm. The control law for each drive is realized in the respective lower-level processor. And since the lower-level processors may work independently from the central upper-level processor, which the algorithm is mainly realized on, master arm's execution of vector $g_d^{(k-1)}$ and realization of the interaction algorithm on the (k-1) step are performed independently one of other. Generally, the $g_d^{(k-1)}$ execution ends significantly earlier than the k^{th} step of the algorithm comes to an end at the central processor.

Further there goes the data exchange with the graphic station. And the initiative come from the module that operates with period 64 ms. The exchange is initiated by transmission of the sync byte (block 1). Till the acknowledgement signal comes the module has a spare time used for computing the vector of current forces and torques $G = (F,M)$ applied to the master arm (block 2). For a higher accuracy the force&torque sensor is sampled 16 times with averaging of data. Upon subtraction of values obtained prior to operation without load vectors of forces F and torques M are generated. The obtained vectors are additionally corrected for possible short kicks of the sensor indications. The process of F and M vectors generation is thoroughly given in description of ZERO SEN command procedure and subprogram for generating force and torque vectors (p.1.2.2. Interim Report # 4 Task 6).

While data (normal unity vector q^{I} and point of contact y_c) from the graphic station are being received the computation goes for a known $g_d^{(k-1)}$ of the Jacobean J and the virtual manipulator's position matrix and, also, of corresponding to it rotation matrix α (block 4), and the position vector of i-th polyhedron approximating the gripper y_i is computed using of (3.7a). The vector y_i is identical to the actual position when the virtual robot has no contact with environment.

The number of received bytes is counted in the process of interrupt handling. Upon data reception the flag is tested for virtual robot's contact with an obstacle imaged by the graphic station complying to the vector $g_d^{(k-1)}$ of generalized coordinates transmitted to the graphic station 1 on the preceding step (block 5). If the flag is set, the proposed position of the virtual robot is checked, one determined by vector $y_i^{(k)}$ and corresponding to vector $g^{(k)}$ of generalized coordinates of the master arm executed on the current step. The vector $y_j^{(k)}$ must satisfy the equation (3.5) if the contact with surface takes place otherwise the contact is lost.

Both in the case of losing contact and absence of flag of contact the virtual robot image in a current position is also to be built on the base of vector of generalized coordinates $g^{(k)}$ without correction and computed vector of master-arm's reaction forces will be zero. [1]

If contact takes place, the matrix C_{k-1} is computed (block 7) with formulas (3.16).

Then vector of the virtual manipulator's generalized coordinates $g_c^{(k)} = g_d + \Delta g_c$ (block 8) is computed with formula (). It is needed for imaging the virtual manipulator on condition of contact with the surface. Also the force F_d^I and torque M_d^I are computed with formulas (3.27) and (3.28). After that, the vector of difference between expected and current reactions $\Delta G = G - G_d$ (block 11) is computed.

Notwithstanding that the corrected vector is fully computed by the module, its constituents ($g_d^{(k-1)}$ and Δg_c) are transmitted to the graphic station separately.

The separate transmission of the correction vector is determined by the fact that the corrected (sliding on the surface) position of the virtual robot may be interpreted as loss of contact with the surface due to an error in computation what will cause the false flag of contact on the next step. For the logical lock of contact the graphic station SW uses the manipulator's vector of generalized coordinates g switching the virtual robot in the state of penetration through the current approximating plane every time when the vector of correction is not zero.

When data exchange with the graphic station is completed, the difference vectors for forces and torques are transformed, respectively, to differential translation $d = (d_x, d_y, d_z)^T$ in the handle's coordinates system and to differential rotation $\delta = (\delta_x, \delta_y, \delta_z)^T$ round the same coordinate axes by means of multiplication by the coefficients that regulate translation and rotation for one step and determine the degree of compliance (block 12). Multiplying the manipulator's current matrix by a transformation matrix composed on the base of differential translation and rotation, the matrix is calculated of the master arm's position for the following step (block 13):

[1] It should be noted that this fact does not imply absence of the next indication of contact because its loss was with the plane approximating the surface on the previous step.

$$T^{(k)} = T^{(k\text{-}1)} \begin{bmatrix} 1 & -\delta_z & \delta_y & dx \\ \delta_z & 1 & -\delta_x & dy \\ -\delta_y & \delta_x & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Basing on the obtained matrix the vector $g^{(k+1)}$ of generalized coordinates is computed (by way of solving a reverse kinematic task) to be executed on the next step of the module's operation (block 14). The components of $g^{(k+1)}$ are the desirable values for the joint drives of master arm. They are controls for the servo drives.

For the software realization of this algorithmic module, further to be referred to as the basic, it needs some additional software means for including it in the control system of manipulator PUMA used as the master arm.

These means are special operational system, which has a title ARPS (Advanced Robot Programming System). It was developed in Russia and improved in the process of work over the project. ARPS was realized on the upper level of SPHERA-36, described in section 4.2.3.

4.2.2 The algorithm and SW prototype of Unit 2 for position/orientation control of mobile double camera for observation of environment

The developed prototype of SW for Unit 2 (system for controlling position/orientation of mobile pair of TV cameras observing work scene) is realized at the upper-level's processor and performs the following operations:

1) Entering an increment in position/orientation coordinates of marks in the reference device fixed on operator's head during interval Δt which corresponds to one cycle of the algorithm's operation.
2) Generating for the above entered data joint coordinates vector of increment in position and orientation matrix of the reference device's marks $\Delta T_{м}$ which must be equal to the increment in position/orientation of the platform
$$\Delta T_{pl} = \Delta T_{м.}.$$
3) Generating current value of the platform's position and orientation matrix in the robot-like device's coordinates system which will be:
$$T_{pl} = T_{pl}^{0}\,\Delta T_{pl} = T_{pl}^{0}\,\Delta T_{м}\,.$$
where T_{pl}^{0} and T_{M}^{0} are initial values of position-and-orientation matrices for the platform and the reference mark device.
4) Generating matrices T_{re} and T_{le} for right and left TV cameras as:
$$T_{re} = T_{pl}T_{corr}^{1} \quad \text{and} \quad T_{le} = T_{pl}\,T_{corr}^{2}\,,$$
where T_{corr}^{1} and T_{corr}^{2} are correcting matrices determining position/orientation of left and right cameras in the platform's coordinates system. It is implied that these cameras' position and orientation conform to those of operator's eyes in the reference device's system of coordinates.
5) Transferring data of T_{re} and T_{le} matrices to graphic stations 1 and 2.
6) Generating the joint coordinates vector of the robot-like device which correspond to matrix T_{pl}.

The lower level's SW generates control data for each of the robot-like device's joint drives which execute target coordinates data produced by the upper level.

Fig.4.9 shows a block diagram of the algorithm for control of position/orientation of the platform bearing the mobile camera pair moved by the robot-like device. A program based on it is realized in operation system ARPS installed at the upper level of the control system in "Sfera-36" unit.

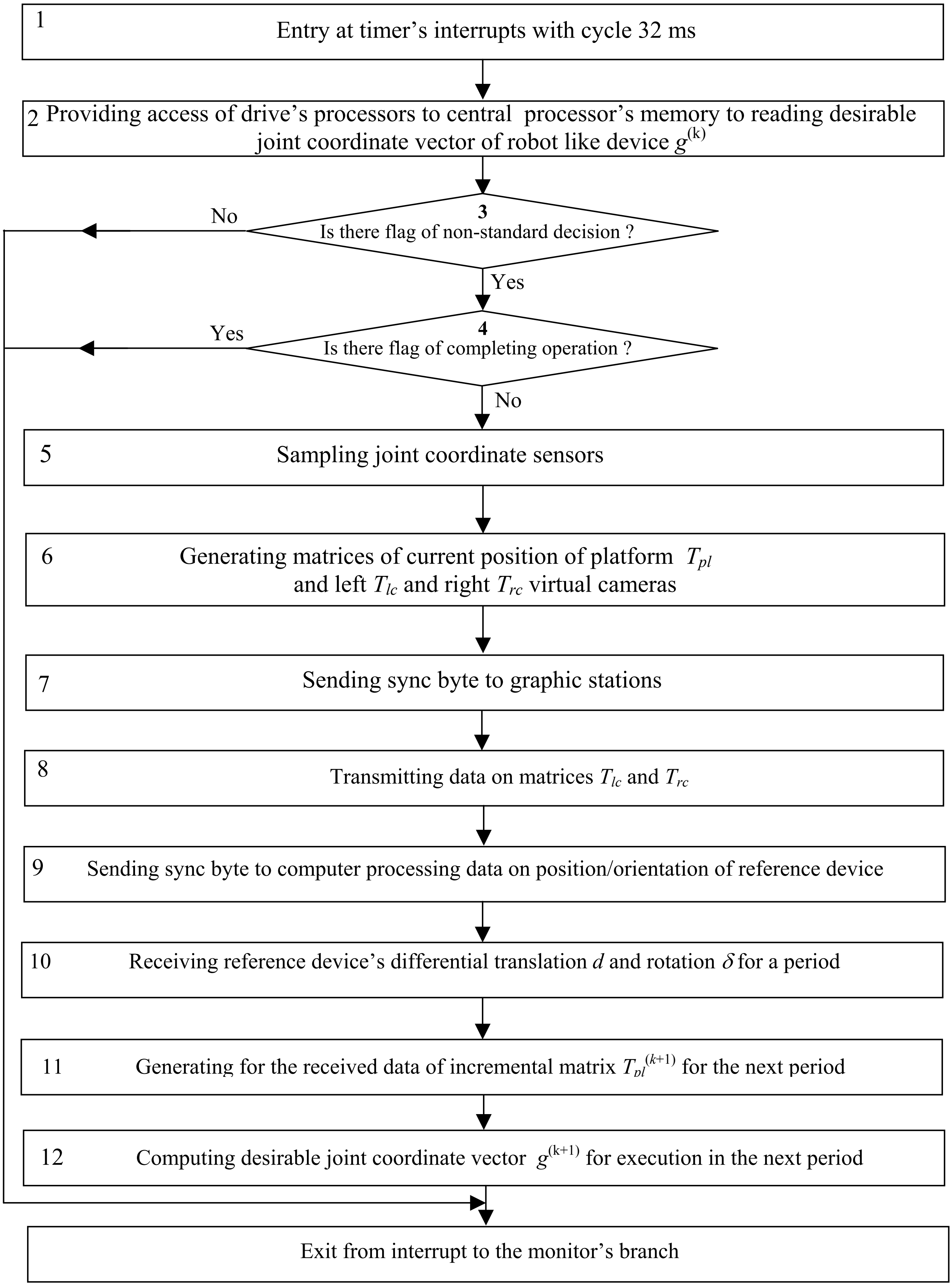


Fig. 4.9 Block diagram of the algorithm of the system for control of video cameras' position and orientation

The algorithm is entered data on the head-borne reference device's position and orientation as increments translation (3 values) and rotation (3 values) relative to axes of the TV camera coordinate system fixed with the camera imaging marks of the reference device. Thus, data volume coming from the computer processing data on the reference device's position/orientation is 6 words, or 12 bytes. The sampling goes every 32 ms determined with a period of the timer in "Sfera-36" control unit. Increments in position ΔX, ΔY, ΔY and orientation δx, δy, δz of the reference device for a period of the timer are to be executed with the platform bearing the observation cameras.

At starting operation of the upper level's cyclic algorithm clocked with the timer, the lower level's drives (unit 2) execute current vector of joint coordinates g^k computed in the previous period of the timer.

Upon that, having completed the check (explained below) of the readiness for permission to proceed (blocks 3 and 4) there goes sampling sensors of joint coordinates and speeds of the robot-like device (block 5) and generating for these data (solving a direct kinematic task) matrix T_{pl} of the platform's current position and orientation and, also, matrices T_{le} and T_{re} of left and right mobile cameras' positions and orientations for transmitting their data to graphic stations 1 and 2 (block 6).

Then, the graphic stations receive a sync byte (block 7) and matrices T_{re} and T_{le} for assigning current positions of virtual cameras (block 8).

Afterwards, the computer processing data on the reference mark device's position/orientation, having received a sync byte, produces increments of reference device's translation dx, dy, dz and rotation δx, δy, δz, for which data increments matrix $\Delta T_{M}^{(k+1)} = \Delta T_{pl}^{(k+1)}$ of the reference device is generated (block 10) having the form:

$$\Delta T_{pl} = \begin{bmatrix} 1 & -\delta_z & \delta_y & dx \\ \delta_z & 1 & -\delta_x & dy \\ -\delta_y & \delta_x & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $d = (d_x, d_y, d_z)^{\mathrm{T}}$ and $\delta = (\delta_x, \delta_y, \delta_z)^{\mathrm{T}}$, respectively, differential translation and rotation relative to axes of the coordinate system bound with the platform bearing the camera pair. Current position of the platform's system relative to the base one on the algorithm's k^{th} step is determined with the position matrix T_{pl}^{k}.

Then, there goes computation of the $(k+1)^{\mathrm{th}}$ desirable point on the platform's trajectory determined with matrix $\mathrm{T}_{pl}^{(k+1)} = \mathrm{T}_{pl}^{(k)} \Delta T_{pl}$ (block 11).

The desirable vector of joint coordinates g_i^{k+1} is computed, then, (block 12), solving a reverse kinematic task for matrix $\mathrm{T}_{pl}^{(k+1)}$, and it is executed with servo drives in the next algorithm's cycle.

The operation system for this algorithm's SW is the same , as in the case of hand's interacting, modified system ARPS described below. For that, the system's language is added to a command HMOVE, which being entered, enables ARPS system to begin and terminate the robot-like device's movement copying that of the head reference device, depending on position of the special external switch.

Upon enabling the mobile cameras' control system and prior to operation (entering HMOVE command) the robot-like device is to be placed in the initial position.

The following additional program modules were created for providing HMOVE command's execution:

- procedure for HMOVE's initialization;
- subprogram for handling interrupts at receiving a byte from the computer processing data on reference device's position/orientation;
- program for executing HMOVE in current cycle.

The initialization procedure first applies to the standard ARPS for computing robot-like device's configuration in the initial position. Next, with the help of a standard subprogram the procedure solves a direct kinematic task for finding initial value of matrix $T_{le}{}^{0}$ determining the platform's position/orientation and then sets the interrupt vector and the flag of non-standard operation for starting a cycle of HMOVE's execution at receiving a byte from the processing computer.

The subprogram handling interrupts is activated at receiving a byte from the data processing computer which is then written to the buffer, the byte counter is incremented and the interrupted program is returned to.

The program executing HMOVE command realizes the basic algorithm shown in Fig.4.9 for exception of the block providing transmission of joint coordinates to drives' processor for execution. This block's operation is provided by standard ARPS.

Upon testing the flag of non-standard operation (operator 3) and ascertaining its being set, the program tests position of the special switch and, if it is on (bit 7 in external register 176616 is cleared) operator 5 is executed.

Receiving data on the reference device's position/orientation from the processing computer (operator 6)is organized with the help of interrupts what enables, while waiting for them, to execute additional computation if a task needs it.

The block diagram of the data reception algorithm is shown in Fig.4.10.

Data in the reception buffer containing 6 words on differential translation and rotation are written to the special array as the transformation matrix ΔT_{pl} for cycle (operator 10).

Upon that, there goes computation (block 11) of $T_{p1}^{(k+1)}$, matrix of the platform's position/orientation and a reverse task is solved for obtaining desirable vector $g^{(k+1)}$ (operator 12). It is done by applying to the appropriate standard ARPS subprograms.

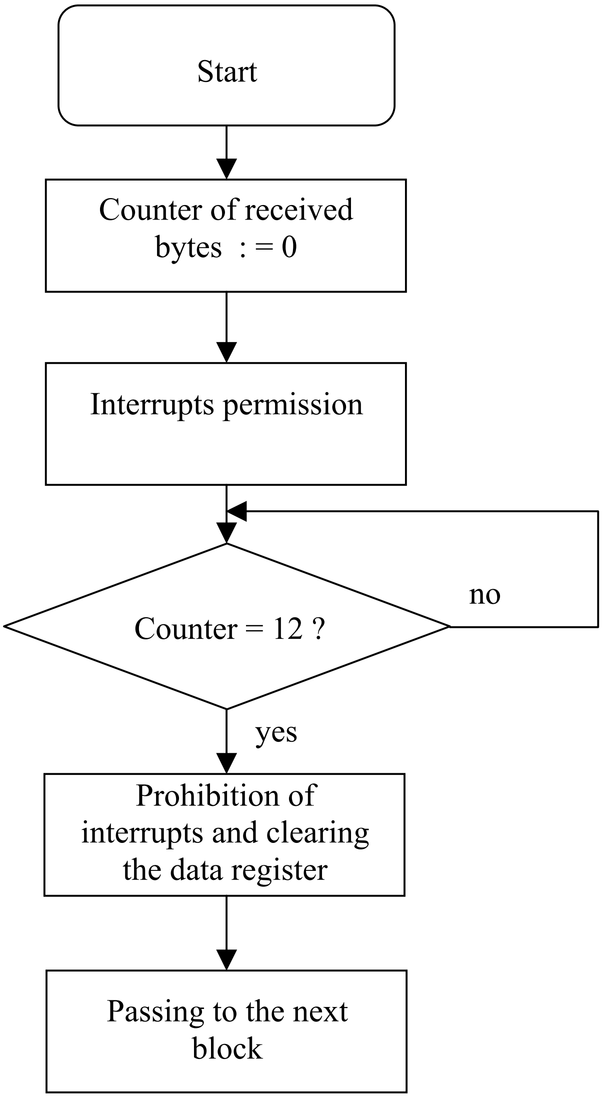


Fig.4.10 Block diagram of operating algorithm for the data reception block.

4.2.3 Special operational system of unit 2 and unit 9

Special operational System which has a title ARPS (Advanced Robot Programming System) - is a programming system developed in Russia and modernized in the process of work over the project. It is basing on a specialized computer and purposed for robot control. With the help of this system setting robot operating modes is accomplished by inputting programs in the computer.

ARPS system comprises the central computer, a video terminal, a floppy-disk drive, a remote hand-control console and input/output lines. Programming is performed by writing control instructions in a special language from the video terminal's keyboard the console is used for teaching robot in programmable positioning points. The input/output lines enable robot's connection with various auxiliary equipment, e.g. with a PC or a force sensor.

ARPS operation system is permanently stored in a programmable ROM that may be exchanged for a read/write memory. In that case the system is easily modified to have several different versions stored on floppy disks.

Upon start ARPS switches into, so-called, monitor mode. This mode enables inputting from the keyboard directives with operations to be executed by robot. The monitor directives enable inputting and editing robot-control program, running and stopping them, teaching robot in points its tool is to go through etc.

The system's being ready for taking monitor directives from operator is displayed by screen messages

>

or

RUN >

Symbol > implies that no program is initialized (robot is stopped). Message **RUN >** means that a program generated by the editor is run.

Robot programs consist of a command sequence that controls robot and checks its execution. A program bears a name of an arbitrary number of symbols, letters A-Z and ciphers 0-9 and point (.) as such.

A program is generated with the help of the editor. Switching into the editing mode begins with a directive

EDIT <name of program>

If name is not given the previously edited program goes by default. If the editor is ready to begin it displays one by one numbers of lines:

1.

2.

etc.

A number displayed, the operator may write commands of two kinds: editor and program commands. Editing is finished by editor command **E**.

ARPS language implies that some operations from the programming point of view are universal, i.e. may be used both as monitor directives and program commands.

The format of directives, of program and editor commands is the same and has the form:

INSTRUCTION argument 1, argument 2...

where

- INSTRUCTION - name of operation which is a sequence of symbols that may consist of one or two parts divided by a blank (e.g. GO and GO READY);
- argument 1, argument 2... distances, speeds, values of angles pertaining to the operation.

If an argument is enclosed in angled brackets it, from the technical point of view, is insignificant and the user may not pay attention to it. In this case, the system uses some default value that is almost always is zero, for example:

MOVE <dx>,<dy>,<dz>

In this case all the arguments are not obliging and therefore a command determining the manipulator movement at 100 mm along axis *x* may be written in such a way:

MOVE 100

or

MOVE 100,0,0

If a program consists of one command it may not be named and initiated by **RUN** directive, it is enough to write (.) before the command. Execution of commands with foregoing points implies that the user program is not initiated (> displayed).

A feature of ARPS system is the possibility of parallel work, i.e. simultaneous preparation and execution of robot control programs when the processor is not used full time. Generally, such a time-sharing is done by a special distributor of resources giving processor time to program modules depending on their current significance. ARPS has no such instrument and sharing is realized by a rather complicated algorithmic structure.

This system, grossly, consists of two branches. The basic branch (monitor's) serves for the dialogue with operator and execution of monitor directives the second branch works with the timer interrupts and serves for execution of robot-control commands and for supporting those monitor directives that use timer interrupts.

Timer interrupts serve two functions. The first one is continuous keeping of a robot current position or a next calculated position. The second one is successive reading of program commands and their execution. The difficulty is that such a mechanism causes appearance of nested interrupts - the interrupt-handling program is interrupted itself. It takes place till execution of the trajectory movement command which stops interrupt nesting and in time that remained after interrupt exit operates the basic branch. Fig. 1.5 shows a simplified block diagram of ARPS timer-interrupt branch.

This diagram does not demonstrate all the complexity of the algorithm but gives a notion of basic logical links appearing by program's joint execution of consecutive and cyclically repeating at every interrupt operators.

For solving tasks of other sorts, to which realization of virtual manipulator's interacting with environment belongs, the standard structure of ARPS operation system and language is insufficient. Therefore a need arises for extending the language introducing additional commands and modernizing the operation system by including interpreting mechanism, executing procedures and blocks for supporting these commands.

In this connection, the diagram shows with dot lines additional program units needed for support of the new commands if the latter use timer interrupts. Unit 1 analyzes type of executed command and triggers a corresponding supporting program. For accomplishing simple and developed tasks it may be sufficient to relate each of them to one monitor directive.

For more complicated tasks requiring permanent perfection it is expedient to use a sufficiently rich set of relatively simple additional commands, which various programs may be composed of, one for debugging is in their number. Therefore a program for accomplishing a complex task is most frequently not optimal one for a number of included commands, because elementary operations of some commands could be performed within others. The additional unit in this latter case is designated with 2.

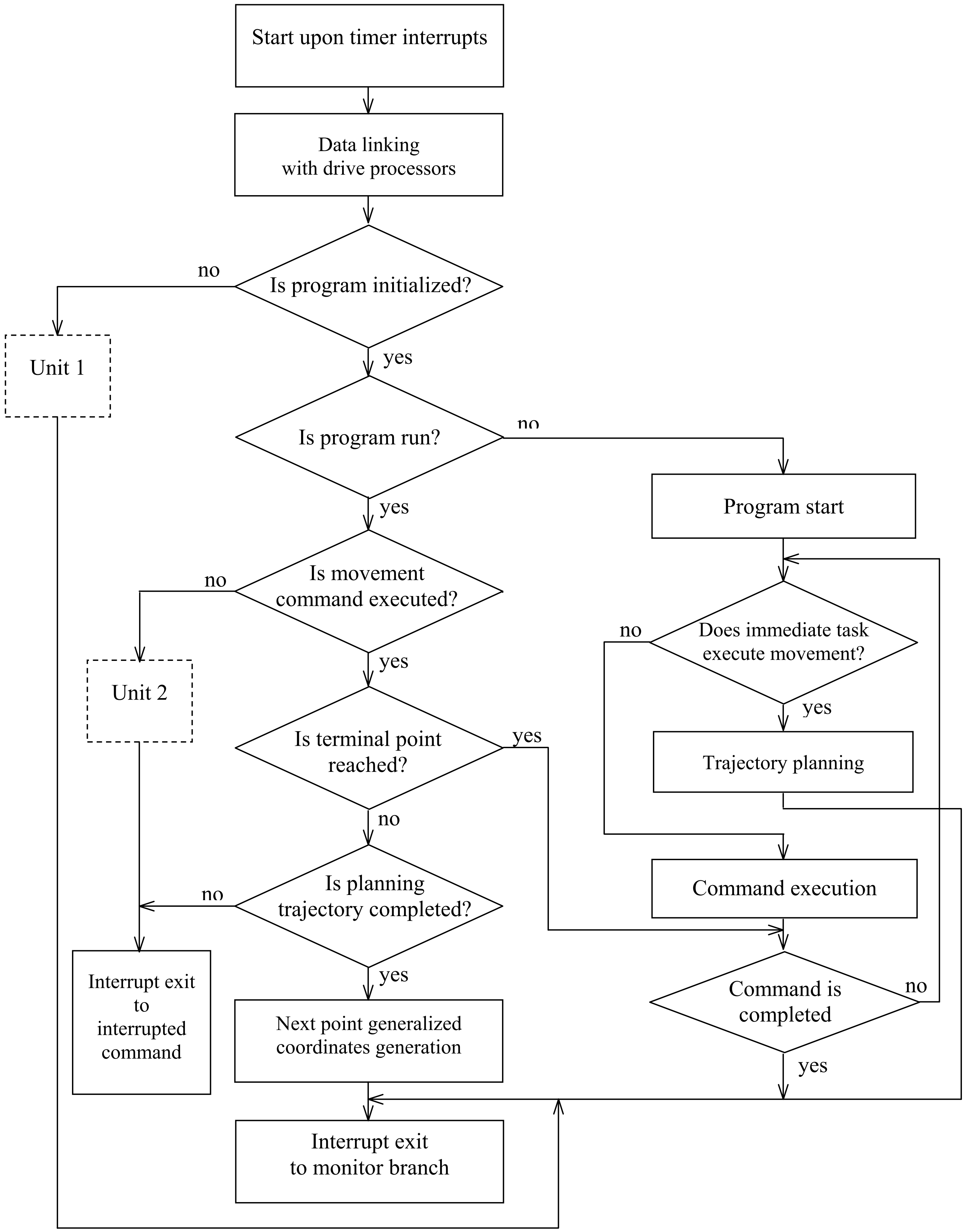


Fig. 1.5 Simplified block-diagram of operation system ARPS operating with timer interrupts (possible extension is shown as units drawn with dot lines)

For interrupting all types of commands of programming system ARPS the operation system uses a universal algorithm that addresses for its work a special table divided in interpreting frames. Each frame (Fig. 1.6) is a definite row of data needed for identification of command and transformation of its arguments.

Every Latin letter is related to a definite sequence of such frames allocated to individual commands.

The system memory holds another table where every Latin letter is related to a base address of the first frame in the sequence. If the language has no/no more commands beginning with a letter the base address is zero.

Command interpretation is performed in the following way. The system finds in the table the base address of the first interpreting frame corresponding to a given command initial. Next the system finds the first frame and analyses its command's type. If this type of command is enabled the name of entered command is compared with the name written in the frame. If identity is none a base address of the next frame is taken from a special field. The next frame found, the operation is repeated. The sequence is tried until a base address of the next frame appears to be zero. The procedure stops and error message is displayed.

Upon finding a frame allocated to the entered command the transformation of arguments is performed by the successive examination of corresponding byte codes. As is seen from Fig. 1.6 the reading of codes is performed upwards in direction of decreasing addresses in the frame. According to a code written in the byte a subprogram is called for transformation of argument in the symbolic form to the binary one or the subprogram for identification of division symbol. The transformation goes until the appearance of the zero byte. If an entered arguments contain wrong symbols and division symbols an error message is displayed.

For the task of human hand's force interaction with virtual robots additional types of data are inputted to the system to be used in computations and as possible arguments for additional commands.

1. Forces

Forces are given in Newton's. The least force value and its range depend on a force&torque sensor employed. For the system developed in the framework of this project they are taken 0,1 N and 100 N, respectively.

2. Torques

Torques is given in [Newton×meter]s. The least torque value and the range also depend on the force&torque sensor employed. In our case they are 0,01 Nm and 10 Nm, respectively.

*3. **Force and torque indexes.***

Indexes are used for allocation of forces and torques to corresponding axes of the tool (handle) coordinate system. For axes x, y, z the indexes are FX, FY, FZ and MX, MY, MZ, respectively.

	Senior bytes (bits 8-15)	Junior bytes (bits 0-7)
Argument sequence direction ↑	Code sequence characterizing argument types (1 in bit 8 means insignificant argument)	0
		Code sequence characterizing division symbols between arguments (generally code 201, or comma)
	Command executing procedure address	
Frame base address	Base address of the next frame (for another command with the same initial) or zero if the initial is exhausted	
	Command special code characterizing its type (bits 1,1,2,3,13,15 are used)	
	Bit	*Command type allocated to a given bit*
	0	Monitor directive enabled upon message >
	1	Monitor directive enabled upon message **RUN>**
	2	Program command
	3	Editing command
	14	Program command interrupting manipulator contour movement
	15	Program command realizing standard manipulator movement
Command name direction ↓	Symbols code sequence making a command name	
	0	

Fig. 1.6 The structure of the interpreting frame

For the operational system's identifying and executing an additional command one needs to generate its individual interpreting frame, a subprogram for transformation of arguments and a procedure for the command execution. As was earlier said, it is needed also to generate a support program for an executive command if it uses the timer interrupts.

Because the system memory space (octal addresses 60000-160000) is entirely filled the additional programs and data are stored in the workspace (addresses 0-60000) where the system writes data resulting from its work to. This space being more than enough its remainder may be used for storing the abovementioned additions (for our case - addresses 40000-60000).

Let us dwell on the generation of a special code for the additional commands that must be written to the interpreting frame. If an additional command can be executed as a monitor directive because of a specific character of the former it need not to be executed in parallel with any program (screen message RUN >). Therefore for additional commands of the monitor-directive type bit 1 in the special code field is always cleared and bit 0 is set.

For an additional program command bit 2 is set in the special code field. If the command is universal one both the bits are set (0 and 2). As for a bit (bit 14) indicating that a command interrupts the contour movement mode, the latter notion is to be cleared.

ARPS programming language includes a mode of contour movement in which a manipulator moves from point to point at constant speed. It does not stop in intermediate points for its movement to the next point begins before it reaches its first destination. In the contour movement all crook points are smoothed, the more that the more a speed of movement, as the centripetal force is constant.

The contour movement is achieved by passing control to the next segment command 0,3 s to the completion of the first one (if the next one is a contour movement command). At the command interrupting contour movement (bit 14 is set, see Fig. 1.6) the manipulator completes execution of the previous command and contour movement stops. The algorithm analyzing the contour movement is very sophisticated one and there is no need in its modernization, therefore the following simple rules are used for setting bit 14 (see Fig. 1.6):

- if execution of the command requires the support of the timer;
- command interrupts the contour movement if time of its execution is long (about 0,3 s).

For additional commands not satisfying the abovementioned conditions bit 14 may be cleared.

Bit 15, set for standard movement commands, must be cleared for all additional commands irrespective of their causing or no manipulator movement.

For calling subprograms that transform arguments in additional programs codes 6(7) and 64(65) are used ones not employed in the standard system version.

For serving the task of human hand's force interaction with a virtual robot the ARPS language is supplied with additional commands which description goes below (a command interrupting the contour movement is labeled with **BREAK**).

ERR ZERO program command or monitor directive

With the help of this program one forms an insensitive zone (zone of possible errors without action of external agents) for correction of forces and torques applied to the handle.

The command format is:

ERR ZERO <force>,<force>,<force>,<force>,<force>,<torque>

where

– force (torque) = minimal absolute value of force (torque) that is regarded as the result of a nonzero collision (#0).

Upon executing this command the system corrects the force&torque sensor data before using them for subsequent computations.

Note: **ERR ZERO** command having no arguments means that the correcting procedure is canceled.

Example:

ERR ZERO 1, 1, 1

it means that all external forces whose absolute value is less than 1N will be considered null.

Generally, in our task the command is used in this very form as the insensitivity zone for torques is enough large by itself.

ZERO SEN a program command or monitor directive

This command is used for reading the force&torque data in a moment when external forces do not act on the handle and, also, for compensation of measured components of gravity forces acting in this position of the handle.

Command format:

ZERO SEN

IFORS program command

This command enables control with force&torque sensor for assigned components of forces and torques.

Command format:

IFORS index, <index >,…,<index>

where

– index = FX, FY, FZ, MX, MY means a component of a force&torque vector assigned for enabled control.

Note: for adjusting specific subtasks a different set of the indexes is used but the finalized program the command enters with the full set of arguments.

Example:

IFORS FX, FY, FZ, MZ

means that force is controlled for all axes of the handle coordinate system and torque - only for Z.

FMOVEL V program command **BREAK**

This command switches on the handle compliant motion taking in account reaction forces in the virtual robot's interacting with environment. The special switch may stop the movement externally.

Command format:

FMOVEL V

FSTOP program command **BREAK**

This command disables further compliant motion and puts the system in the standard mode.

Note: while dealing with a real obstacle this command executes the transient process to the null reaction of the obstacle.

Summing up, the following basic kinds of program structures needed for ARPS modernization may be distinguished:

- subprograms for transformation of arguments in additional commands;
- procedures for executing additional command;
- programs for supporting additional command;
- auxiliary subprograms.

Consider in more detail the additional program structures for realizing the task of human hand's reactant interaction with the virtual robot.

Subprogram for transformation of forces and torques

This SP performs transformation of force and torque values in symbolic form into the binary one and writes the transformed values to R0 register as whole numbers in the range 1-10000 (correspond to the range of inputted values 0,01-100). The subprogram is called with code 6(7).

Subprogram for transformation of force and torque indices

This SP analyzes the symbolic expression of indices and relates each of them to a definite number in the range 1-6 ($F_x \to 1$, $F_y \to 2$, $F_z \to 3$, $M_x \to 4$, $M_y \to 5$, $M_z \to 6$), and writes it to R0 register. The SP is called with code 64 (65).

It should be noted that a sequence of transformed arguments for each additional command is formed by the standard interpreting algorithm and is written to the common work memory space (0-40000). The standard program, executing algorithm in ARPS language, calls a procedure for execution of an immediate command in such a way that the argument sequence address will be written to register R4.

Subprogram for taking force&torque sensor data

This SP is a auxiliary one and reads data via standard input/output lines. The sensor measures six values of forces and torques, that we designate as f_n, f_s, f_a, m_n, m_s, m_a. The non-acted sensor gives nonzero readings, null offsets that will change with time. Therefore the periodic calibration is expedient for read-out of null offsets to be compensated for. For better accuracy the SP reads the data 16 times, averages them and writes to a special data array.

Subprogram for determining handle weight vector components in the tool (handle) coordinate system

To compensate for null offsets caused by gravity vector p, given in the base system of coordinates, its components must be determined in the tool (handle) coordinate system.

Knowing a matrix of rotation nsa of the handle coordinate system relative to the base system of coordinates xyz $\begin{vmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{vmatrix}$ the SP computes the components f_{pn}, f_{ps}, f_{pa}, m_{pn}, m_{ps}, m_{pa} of the gravity vector in the tool coordinate system with expressions:

$f_{pn} = n_z p$,

$f_{ps} = s_z p$,

$f_{pa} = a_z p$,

$m_{pn} = r f_{ps}$,

$m_{ps} = r f_{pn}$,

$m_{pa} = 0$,

where p - gravity vector in the base system of coordinates (total handle-with-sensor weight is determined beforehand and periodically corrected);

r - distance between the center of the sensor and the common center of mass (handle with sensor). It is assumed that both the centers lie on a axis of the tool system of coordinates.

ZERO SEN command execution procedure

The procedure gets start only when the handle is free from external forces. The procedure addresses the SP reading the force&torque sensor and gets as the result components of forces and torques $f_n^0, f_s^0, f_a^0, m_n^0, m_s^0, m_a^0$ in the tool coordinate system (superscript 0 means that the component are got in absence of external forces and torques) and stores them in a special data array. Then the procedure addresses SP for getting components of the gravity vector $f_{pn}^0, f_{ps}^0, f_{pa}^0, m_{pn}^0, m_{ps}^0, m_{pa}^0$, and also stores them. The quantization step of the force&torque sensor is rather large and different for different components (1 division ≈0,3 N), moreover, computations are made in the fixed-point format giving some additional error. Therefore this procedure is better to be performed with the handle vertically up or down, i.e. along axis z of the base system of coordinates. In this case all weight will be concentrated in f_{pa}^0 component, the other ones being null.

ERR ZERO command execution procedure

This procedure writes values of forces and torques inputted by operator to a special memory space for subsequent application (their respective designations are $F_n^0, F_s^0, F_a^0, M_n^0, M_s^0, M_a^0$).

Subprogram for computation of forces and torques applied to the handle

This SP is used by the basic algorithm for computation of force F and torque M vectors applied to the handle and measured with the force&torque sensor. The SP addresses SP for the strain force&torque reading and then SP for the gravity vector decomposition. Basing on acquired data $f_n, f_s, f_a, m_n, m_s, m_a, f_{pn}, f_{ps}, f_{pa}, m_{pn}, m_{ps}, m_{pa}$ components are computed of vectors F and M in the tool coordinate system (only F_n and M_n are given for an example):

$$F_n = f_n - f_{pn} - (f_n^0 - f_{pn}^0),$$

$$M_n = m_n - m_{pn} - (m_n^0 - m_{pn}^0).$$

For the rest components formulas are similar.

These formulas show that for getting net values of applied forces and torques one needs subtract from weight-free sensor readings with external forces applied those with of external forces removed. If the control is executed only for forces the rotation matrix and, hence, components of weight do not change. In this case $f_{pn} = f_{pn}^0$ etc. what simplifies the formulas.

In reading the force&torque sensor some little failures are possible that are caused by interference of various nature (the sensor indicates an external collision that is really none). Therefore SP operates with minimal values of forces and torques that may be attributed to external actions. The more little values are considered as noise. Hence, the final values of F and M (only F_n and M_n for an example) are screened on the following principle:

F_n is saved if $|F_n| \geq |F_n^0|$, $\quad$ $F_n = 0$, if $|F_n| < |F_n^0|$

M_n is saved if $|M_n| \geq |M_n^0|$, $\quad$ $M_n = 0$, if $|M_n| < |M_n^0|$.

Subprogram for interrupts handling in data exchange with the graphic station

This auxiliary SP operates in the moment of reception of a successive data byte from the graphic station. It writes a received byte out of the external register to a special data array allocated to the graphic station and increments the byte counter.

Subprogram for computing the Jacobean and the matrix of virtual robot position

For computing the Jacobean in the tool coordinate system of the virtual robot this SP applies a method proposed in work [27].

The way of realizing this method makes possible both acceptable speed of computation and adaptability to the task.

For computation of the last three Jacobean vector - columns $J_4(g)$, $J_5(g)$, $J_6(g)$ enough simple analytic expressions are used:

$$J_4(g) = \begin{bmatrix} d_6 S_5 S_6 \\ d_6 S_5 C_6 \\ 0 \\ -S_5 C_6 \\ S_5 C_6 \\ C_5 \end{bmatrix}, \quad J_5(g) = \begin{bmatrix} d_6 C_6 \\ -d_6 S_6 \\ 0 \\ S_6 \\ C_6 \\ 0 \end{bmatrix}, \quad J_6(g) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

For the first tree Jacobean columns analytical expressions are also available but are too bulky and badly suitable for program realization. Therefore the first three columns $J_j(g)$ are computed in a regular way out of blocks of matrixes

$$U_j(g) = \prod_{i=j}^{6} {}^{i-1}A_i \ ,$$

where ${}^{i-1}A_i$ - matrix of position, relative to the $(i\text{-}1)^{\text{th}}$ link coordinate system, of the i-th link coordinate system.

If represent matrix U_j as:

$$U_j = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

then the corresponding Jacobean column $J_j(g)$ is computed with a formula:

$$J_j(g) = \begin{bmatrix} p_x n_y - p_y n_x \\ p_x s_y - p_y s_x \\ p_x a_y - p_y a_x \\ n_z \\ s_z \\ a_z \end{bmatrix}.$$

Matrix $U_4 = {}^3A_4\,{}^4A_5\,{}^5A_6$ is computed analytically for its being enough simple and matrix U_3, U_2, U_1 by way of the numerical multiplication with PC:

$U_3 = {}^2A_3 U_4$, $U_2 = {}^1A_2 U_3$, $U_1 = {}^0A_1 U_2$.

Following this way the matrix of virtual robot position appears a by-product:

$T = U_1 = {}^0A_1\,{}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5\,{}^5A_6$.

Below the matrices are given that are input data for computation of the first three Jacobean columns:

$${}^0A_1 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},\quad {}^1A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix},\quad {}^0A_1 = \begin{bmatrix} C_3 & 0 & S_3 & a_3C_3 \\ S_3 & 0 & -C_3 & a_3S_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$U_4 = \begin{bmatrix} C_4C_5C_6 - S_4S_6 & -C_4C_5S_6 - S_4C_6 & C_4S_5 & d_6C_4S_5 \\ S_4C_5C_6 + C_4S_6 & -S_4C_5S_6 + C_4C_6 & S_4S_5 & d_6S_4S_5 \\ -S_5C_6 & S_5S_6 & C_5 & d_6C_5 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Here as equally in the formulas for computation of the last three Jacobean columns, the following designation are used:

$S_i = \sin g_i$, $C_i = \cos g_i$, $S_{ij} = \sin(g_i + g_i)$, $C_{ij} = \cos(g_i + g_j)$,

d_i, a_i - parameters of joints in the virtual robot.

IFORS command execution procedure

This procedure generates the frame of 6 words, logically related to forces and torques. If the word contains 1 that means that this force (torque) is controlled and zero means absence of control. Complying with the set of numbers 1..6 formed by interpreting pointers of forces and torques FX, FY, FZ, MX, MY, MZ the procedure finds corresponding words in the frame and sets (writes 1) their least bit. Besides the procedure generates the general flag of the force-torque control.

FMOVEL V program executing procedure

This procedure actually only initializes the compliant motion of the handle and waits for its completion, i.e. marks the initial and final cycles of the basic algorithm module's operation. For this purpose the procedure sets the general flag of non-standard mode of operation and orders a number of program supporting the task of human hand's interaction with the virtual robot. Choice of a support program is realized in the block 2 of the timer interrupts handling branch.

Next, the procedure waits for the moment of the special switch's turn-off (setting of bit 7 in external register 176616) upon what clears the flag of the non-standard mode of operation.

4.2.4 The algorithms and SW of graphic stations

4.2.4.1 The algorithms and SW prototype for generation the augmented reality image

The basic functions of SW installed at Graphic Station 1 (GS1) are the following:

- generating video image of the actual environment “augmented” with the virtual manipulator’s image;
- generating data needed for realizing force interaction of the virtual manipulator with environment, flag of contact and coordinates of point the virtual manipulator contacts obstacles of environment in and, also, forming the unity normal to the surface in the point of contact.

The basic SW functions realized at GS2 are the same as those of GS1 except that it serves the right camera.

Besides that, this GS2 has no SW realizing the virtual manipulator’s interaction with environment.

Generating the “augmented” image comprises the following operators:

- generating the virtual object's geometric model (that of the anthropomorphic space manipulator) and also geometric model of environment (Orbital Station surface with objects transported by the manipulator) for data entered by the operator beforehand;
- entry of coordinates for position and orientation of the left observation camera and computation of those for the right one;
- entry of master arms current joint coordinates used for acquisition of the virtual manipulator's current state;
- generating the virtual manipulator's computer image and geometric model of environment which attitude and scale correspond to current position and orientation of the left (right) camera;
- entry (capture) of image of remote actual environment obtained with the left (right) camera of Unit 1;
- registering geometric model images of the virtual manipulator and environment with the image of actual environment obtained with the help of the left (right) camera supplying the screened fragments of environment for the screening parts of the manipulator, viz. realization of actual environment "augmented" with virtual objects;
- outputting the image corresponding to the left (right) camera to the left (right) display.

The general algorithm for generating the augmented image, which block diagram is shown in Fig.4.11, operates cyclically:

1. Clearing the Frame Buffer containing data on video image of environment, viz. colour and brightness of each screen point (pixel) displayed on the screen.
2. Upon checking and handling possible events from the queue (clicks of the mouse and keyboard) acquisition of coordinates is executed for position and orientation of operator's head and, also, the virtual manipulator's joint coordinates.
3. Entry of coordinates for position/orintation of the video camera and also entry of master arm's joint current coordinates.
4. Generating image of geometric model of environment, viz. augmenting the Frame Buffer and filling, so-called, ZBuffer containing "depths" of screen elements (pixels), i.e. data on distances to the observation cameras.
5. Capture of the video frame and copying fdom video capture buffer to the Frame Buffer a "captured" (with the help of the video capture board) video frame of actual environment.
6. Forming an image of the actual remote environment augmented with that of the virtual manipulator, i.e. substituting in the Frame Buffer those elements of

environment's image (pixels) for the corresponding ones of the virtual manipulator that meet the following conditions. The first – pixel's belonging to the both images (environment's and manipulator's) and the second – the "depth" of the manipulator's substituting element is less than that of actual environment. The second condition is tested with data stored in the ZBuffer.

7. Displaying the augmented image of actual environment, i.e. copying data from the Frame Buffer.
8. Return to the first step.

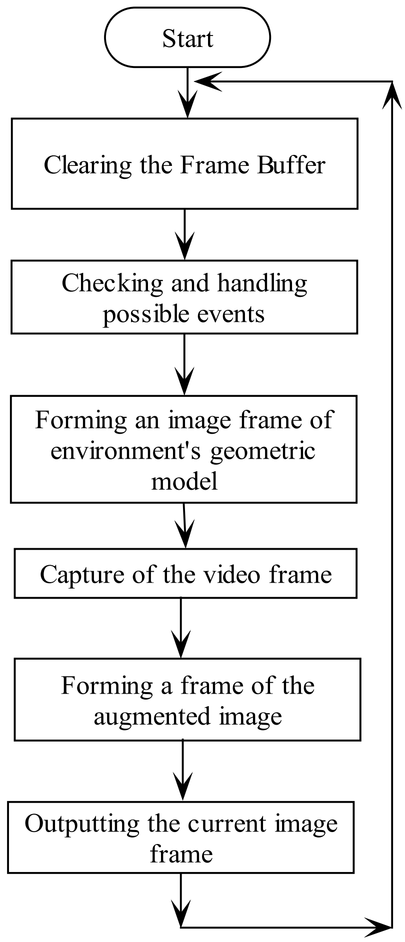


Fig.4.11 General block diagram of the algorithm for generating image of Augmented Reality.

This part of GS' SW has three streams. The first stream, properly, realizes all basic functions needed for forming the augmented image the second streem provides reception the position/orientation TV camera wordinotes the third stream forms the image of environment, viz. recording to the Frame Buffer digitized image of the actual environment obtained with the observation cameras. Note, that generating the data needed for the force interaction of the virtual manipulator with environment is allocated to the third SW stream described, as was mentioned, in the previous Report.

The language of the first stream is Pascal, the compilator Free Pascal 1.07.

The language of the second and third stream is C^{++}, the compilator Microsoft Visual C^{++} 6.0.

The operational system is Windows 2000 Profeccional. Contents of catalogues and files of the first GS SW streamand also the detailed description of each file are given below.

4.2.4.1.1 Contents of catalogues and files

Catalogue Src/Core

Config.pas

- contains a procedure for loading the configuration file LoadConfiguration()
- this procedure is called at the program start from a procedure InitGlobals in Globals.pas
- it assigns values

Consts.pas

- here variables used by initialization and in the process of operation are declared:

1. ScrResX/ScrResY – screen resolution or window size
2. FullScreen - full screen operation if True
3. StatusEnabled - a window with debugging data is displayed if True and operation mode is windowing,
4. RSPort - textual constant with a name of COM-port through which the camera coordinates come.
5. ManipulatorPosX,ManipulatorPosY,ManipulatorPosZ ManipulatorAngleX,ManipulatorAngleY,ManipulatorAngleZ – manipulator's position and orientation in the base system of coordinates
6. EnvScaleX,EnvScaleY,EnvScaleZ – scale coefficients of the environment
7. ManScaleX,ManScaleY,ManScaleZ - scale coefficients of the manipulator

These variables' values are loaded from a file named ConfigurationFileName

If the file has syntactic errors the variables' values are assigned by default

EventMan.pas

- description of a class tEventManager responsible for handling events coming from the keyboard and mouse and, also, for data acquisition thought COM-port.

The class' instance is declared in Globals.pas and is created in procedure InitGlobals() at work start.

Globals.pas

- declares global variables and procedures InitGlobals()/DisposeGlobals()

InitGlobals() create objects EventMananger,GLRenderer,CPUTimer and StatusWindowManager

DisposeGlobals() destructs created objects upon program completion

Logger.pas

- contains functions for writing debugging data to external file

1. Function InitLogger() opens a file for writing, CloseLogger() finishes work with a file.
2. Functions PushProc(<procedure name>)/PorProc() write/clear calling sequence stack (controls nested procedures).
3. Function Fatal() emergently ends program operation saving an error message
4. Functions Log(<string>)/LogF(<number>) writes string/number to the file of debugging messages status

StatusW.pas

- registering and displaying window with debugging data(Status Window)

This module describes a class tStatusWindowManager, which instance is initialized in a module Globals.pas (procedure InitGlobals) at program start.

Debugging data window is accessible only in a window mode at a value of variable StatusEnabled = True.

Timing.pas

- timing system, describes class tCPUTimer

<u>Catalogue Src/Headers</u>

B_Bitmap.pas

- a module for work with raster images + their loading/saving in files .BMP

L_Stream.pas

- abstract interface of entry/exit stream (tStream) and realization of work with files (tFileStream)

L_Types.pas

- definition of basic data types(Lfloat/tVector3/tMatrix3/tMatrix4)

<u>Catalogue Src/Include</u>

Defs.inc

- definitions for preprocessor

Catalogue Src/Loaders

L_Parser.pas

- simple syntactic analyzer of input symbol stream (used for reading configuration file)

Catalogue Src/OpenGL (standard headers of OpenGL)

L_GL.pas - basic OGL functions

L_GL32.pas - WGL (Win 32 OGL interface)

L_GLU.pas - Open GL utility functions

Catalogue Src/Renderer

E_Env.pas

E_Man.pas

E_Obj.pas

- description of abstract class tRenderObject

GL_Setup.pas

- auxiliary procedures used for initialization of OpenGL and for setting parameters for displaying images (matter and lighting).

Man_Obj.pas

- describes class tManipulatorModel realizing functions of construction of the manipulator's model and contains parameters of this model

R_Render.pas

- describes class tGLRenderer realizing functions of construction of virtual images and registration them with video

V_CapVidStream.pas

- describes class tCaptureStream realizing the capture interface

Class tCaptureVideoStream is a successor of abstract class tVideoStream, it realizes the video capture using a dynamic library «dshowtest.dll».

The class contains the following procedures and functions in section "public":

constructor Init(parentWin:HWnd)

function getVideoSize(width,height:pinteger):bool

Function GetNextFrameM(PTR:pbyte):integer;Virtual;

The constructor of Init(parentWin:HWnd) initializes and triggers the video capture by calling function InitCapture out of dynamic library dshowtest.dll.

Function getVideoSize(width,height:pinteger):bool serves for video capture of frame created in getImageSize out of library dynamic dshowtest.dll.

Function GetNextFrmeM(ptr:pbyte):integer;Virtual serves for video capture of a frame created in dshowtest.dll library and for writing this frame to the frame's intermediate buffer for what this function calls getImage function out of this library.

V_VidStr.pas

- describes an abstract class with interface used for obtaining video images

Instance of tCaptureVideoStream class(successor of tVideoStream) is used for obtaining video image.

vfw.pas

- standard interface Video For Windows (used in V_CapVidStream.pas)

win32aux.pas

- contains a standard windowing function for the main window of program.

Called out indirectly in the main program cycle (main.pas).

Uses objects EventMan and StatusWindowManager

<u>Catalogue Src/Utils</u>

LongStr.pas

- class tLongStr realizing operation with long strings

this class is used for generating debugging messages in Logger.pas module

ShellAPI.pas

- standard headers of functions ShellAPI (part of Windows)

Utils.pas

- various functions for transformation of data

./main.pas

- the main program module, calls initialization and organizes handling of events with means of WinAPI

4.2.4.1.2 Program structure

Program run-time consists of event response, data receiving using RS232 ports and rendering of the real environment augmented by the virtual manipulator.

The whole system consists of the following interacting objects:

- EventManager – mouse/keyboard event handler and receiver of the data from RS232 port using RSExchange.dll library.
- GLRenderer – manager of the rendering process and OpenGL setup.
- KeyMapper,CommandServer,VarsManager – the implementation of internal command interpreter , configuration variables manager and key press/release event handler.
- GraphicsWindowManager and StatusWindowManager – window managers for debug information output
- CPUTimer – precise timing unit

GLRenderer object contains additional subobjects which implement the following subsystems :

1. VideoStream object for video capture using 'dshowtest.dll' library
2. ManipulatorObject for environment geometrical model and virtual manipulator rendering

All objects are initialised in InitGlobals() procedure when the program starts and are destroyed in the DisposeGlobals() procedure.

Unit main.pas consists of InitGlobals() call and organisation of the event handling loop. After program termination procedure DisposeGlobals() is called.

<u>Unit R_Render.pas</u>

This unit implements tGLRenderer class which manages main rendering window and controls virtual camera.

Description of tGLRenderer class

Fields :

- UsingMatrix:Boolean – if RS232 port is used as an information source
- TransformMatrix:tMatrix4 – 4x4 transformtation matrix from RS232 port
- Width,Height:LongInt – main window width and height
- CamWidth,CamHeight:Lfloat – projection viewport parameters
- StereoRendering:Boolean – if stereo mode is used
- InverseTransformationOrder:Boolean – transformtation order ("translate-rotate" or "rotate-translate")
- CameraX,CameraY,CameraZ:Lfloat – camera coordinates
- CameraAngleX,CameraAngleY,CameraAngleZ:Lfloat – camera angles
- CamZNear,CamZFar:Lfloat – coordinates of near and far cut planes
- CamAspectRatio:Lfloat – screen aspect ratio
- ShotIndex:LongInt – file number for screen shots
- CamMatrix1,CamMatrix2:Array[1..16] Of Double – transformation matrices for left/right eye
- ProjectionMatrix,CameraTransformMatrix:Array[1..16] Of Double – transformation matrices

- ManipulatorObject:pEnvironmentModel – an object which implements manipulator control and environment rendering
- VideoStream:pVideoStream – an object which implements video capture interfacepVideoImage:pbyte – pointer to the memory block with current video frame

Methods :

- Constructor Init(Title:String;W,H:LongInt;Bits:LongInt;StereoMode:Boolean) – initialisation of main program window
- Procedure SetCameraPosV(pos:tVector3) – set camera position
- Procedure SetCameraAngleV(angle:tVector3)- set camera angles
- Procedure SetCameraPosF(X,Y,Z:Lfloat) – set camera position
- Procedure SetCameraAngleF(AX,AY,AZ:Lfloat) – set camera angles
- Procedure UpdateCamera – rebuild transformation matrices when camera moves
- Procedure RegisterCommands – register some internal command for interpreter
- Procedure SetInitialProjectionValues – set some projection values(viewport sizes , aspect ratio etc.)
- Procedure MakeCameraTransform – build transformation matrices
- Procedure Clear – clear the viewport
- Procedure RenderAll – build the image of environment and manipulator
- Function GetScreenShotImage:pBitmap – get current screen image
- Procedure MoveCamera(HowMuch:Lfloat);
- Procedure ShiftCamera(dX,dY,dZ:Lfloat) – shift camera parallel to coordinate axes
- Procedure ChangeAngles(dAX,dAY,dAZ:Lfloat) – change camera angles
- Procedure SetCameraTransformation – load camera transformation matrices to OpenGL
- Procedure SetCameraTransformation_Left – set left eye's transformation matrix
- Procedure SetCameraTransformation_Right – set right eye's transformation matrix
- Procedure SetProjectionOnly – load projection matrix to OpenGL
- Procedure MakeScreenShotC(Param:String) – save screen shot
- Procedure MoveCameraC(Param:String) – move camera in the specified direction
- Procedure ShiftCameraC(Param:String) – shift camera position
- Procedure ChangeAnglesC(Param:String) – change camera angles
- Procedure SetCameraPosC(Param:String) – set camera position
- Procedure SetCameraAngleC(Param:String) – set camera angles
- Procedure SwitchTransformationOrderC(Params:String) – switch transformation order ("translate-rotate" or "rotate-translate")
- Procedure SwitchMatrixInputC(Params:String) – switch the coordinates input mode (RS232 port or keyboard)
- Destructor Done - destructor

<u>Unit Collision.pas</u>

This unit contains procedures for dynamic collision detection between virtual manipulator and environment model.

Procedures and functions:

- Function FindCollision:Boolean – detects a collision between manipulator and environment and sets the collision flag.
- Procedure GetRSFMHandleData – data conversion from RS232 port into internal format

Unit Commands.pas

This unit implements the class tCommandServer which is a simple command interpreter of the internal language.

Constants :

MAX_COMMANDS_IN_SERVER = 1000 – maximum number of registered commands
MAX_DEPTH_IN_SCRIPT = 10 – maximum number of embedded configuration files

Data types:

tCommandProc = Procedure(Param:String) Of Object – a pointer to the command executor procedure.

The declaration of tCommandServer class.

Fields:

- NumCommands:Longint – number of registered commands
- CmdList – list of commands
- EmbeddedDepth – depth of configuration files embedding which is used to prevent circular references.

Methods:

- Constructor Init – constructor which registers two commands - 'ExecuteScript' and 'Quit'
- Function FindCommand(CMD:String):LongInt – command search in the list
- Procedure RegisterCommand(CMD:String;Executor:tCommandProc) – registration of a new command. CMD – command name, Executor – a pointer to the executor procedure.
- Procedure UnregisterCommand(CMD:String) – removes a command from the list
- Procedure ExecuteCommand(CommandString:String) – execution of CommandString
- Procedure ExecuteScriptC(Param:String) – execution of external file
- Procedure QuitC(Param:String) – quit the program
- Destructor Done – destructor

Unit Consts.pas

This unit contains declaration of global constants and variables which are used during a run-time. These variables can be modified using the internal command interpreter.

Constants and variables:

- WindowTitle = 'Test OpenGL Application' – main window title

- ScrResX,ScrResY:Lfloat – window width and height
- XbitmapOfs,YBitmapOfs:Lfloat – video image offset
- FullScreen:Lfloat = 0.0 – full screen mode flag
- StereoEnabled:Lfloat = 0.0 – stereo mode flag
- ManipulatorPosX,ManipulatorPosY,ManipulatorPosZ: Lfloat – the coordinates of virtual manipulator
- ManipulatorAngleX,ManipulatorAngleY,ManipulatorAngleZ: Lfloat – the orientation of virtual manipulator
- EnvScaleX,EnvScaleY,EnvScaleZ: Lfloat – environment scaling factors with respect to axii Ox/Oy/Oz
- ManScaleX,ManScaleY,ManScaleZ: Lfloat – manipulator scale with respect to axii Ox/Oy/Oz
- CAMFOVY:Lfloat – camera field of view
- FovStepSize:Lfloat – the increment step of FOV
- CameraStepSize:Lfloat = 0.01 – the increment step of camera position

Unit Correct.pas

This unit contains a procedure whic corrects the data received from RS232 port

Procedure:

- Procedure CorrectData(RMIn:tMatrix3;Vin:tVector3;Var RMOut:tMatrix3;Var Vout:tVector3) – input data correction (orientation matrix and camera position)

Unit EventMan.pas

This unit implements tEventManager class which handles mouse/keyboard events and receives data from RS232 port.

The declaration of tEventManager class

Fields :

- WindowIsActive:Boolean – is the main widow active
- MouseEnabled:Boolean – mouse tracking flag

Methods :

- Constructor Init – constructor
- Procedure Idle – procedure which is called every frame
- Procedure MouseMove – mouse movement tracking
- Procedure MouseLUp(X,Y:LongInt) – left button press event handler
- Procedure MouseLDown(X,Y:LongInt) – left button release event handler
- Procedure MouseRUp(X,Y:LongInt) – right button press event handler
- Procedure MouseRDown(X,Y:LongInt) – right button release event handler
- Procedure InitRS232 – RS232 port initialisation
- Function RS232Receive:integer – the function which is called from the data receiving thread
- Destructor Done - destructor

Unit KeyMapper.pas

This unit implements tKeyMap class which is responsible for binding of internal commands to key press events. Also it contains some virtual key codes which are not declared in standart Windows headers

The description of tKeyMap class.

Fields:

Keys:Array[0..255] Of Boolean – array of ‘keypressed’ flags
Bindings:Array[0..255] Of Record NumCommands:LongInt; Commands:Array Of String;End – the list of ‘KEY’<->’command’ bindings

Methods:

- Constructor Init - constructor
- Procedure BindC(Param:String) – binding of specified command to some key press event
- Procedure UnBindC(Param:String) – cancel binding
- Function GetKeyIndex(KeyCode:String):Byte – search of the command in the list
- Procedure SendKey(Key:Byte;KeyState:Boolean) – call binded command
- Destructor Done - destructor

Unit E_Man.pas

This unit implements the tEnvironmentModel which manages the rendering of the environment and manipulator.

The declaration of tEnvironmentModel class.

Methods:

- Constructor Init - constructor
- Procedure RenderAll – the rendering of environment model
- Procedure SelectJoint(JIndex:LongInt) – manipulator joint selection
- Function GetJoint:LongInt – select joint index
- Procedure SetJointAngle(JointIndex:LongInt;JointAngle:Lfloat) – set current joint angle value
- Function GetJointAngle(JointIndex:LongInt):Lfloat – get current joint angle value
- Procedure SetBackgroundImage(SrcPtr:Pointer) – set current background image (video frame)
- Function GetObjectCount:LongInt – return the number of environment objects
- Function GetObject(Index:LongInt):pRenderObject – get environment object
- Destructor Done - destructor

Unit E_Obj.pas

This unit declares an abstract class tRenderObject which is an interface for environment object rendering. Successors of tRenderObject are concrete environment objects.

The declaration of tRenderObject class.

Methods:

- Constructor Init – initialisation and internal list preparation
- Procedure Render – rendering of the object
- Function GetBoundingVolume:tBoundingVolume – return the object parameters for collision detector
- Destructor Done – destructor

Unit GL_Setup.pas

This unit contains common OpenGL operation mode setup.

Procedures:

- Procedure InitOpenGL – loading opengl32.dll library
- Procedure SetupZBuffer – Z-buffer enabling
- Procedure SetupLighting – lighting setup
- Procedure PrepareGLLighting – light sources setup
- Procedure PutGLMaterial(MaterialType:Byte) – object material selection

Unit L_Parser.pas

This unit implements a simple lexixal analyser - tParser which is used for reading configuration files.

Constants:

- MAX_STRINGS_IN_TEXT_FILE = 1000 – maximum number of text string in the input stream

Loken types:

- TOKEN_UNKNOWN = 0 – unknown symbol
- TOKEN_DELIMITER = 1
- TOKEN_STRING = 2
- TOKEN_NUMBER = 3
- TOKEN_CHAR = 4
- TOKEN_EOLN = 5 – end of line flag
- TOKEN_EOF = 6 – end of file flag

The description of tParser class:

Fields:

- NumStrings : LongWord – number of string in the input stream
- Strings : Array[1..MAX_STRINGS_IN_TEXT_FILE] of String – array of read strings

- StreamLen : LongInt – stream length in bytes
- Position : LongInt – current stream position
- ReadBuffer : pTempCharArray – auxillary buffer
- TokenType : LongInt – current token type
- Token : String – current token

Methods:

- Constructor Init(InStream:pStream) – constructor which reads data from the input stream
- Constructor InitFromFile(FileName:String) – constructor which reads the data from external file
- Procedure LoadStream(InStream:pStream) – loading of stream
- Procedure DecodeStream – conversion of 'Strings' array to 'ReadBuffer'
- Procedure AddCharToStream(c:char) – add a symbol into the internal buffer
- Procedure Rewind(n:LongInt) – return n symbols into the stream
- Procedure FFwd(n:LongInt) – skip n simbols
- Function NextChar:char – reads next symbol from the stream
- Procedure SkipString – skips all symbols until the end of current line
- Procedure SkipSpaces – skips spaces
- Procedure SkipComments – skips comments
- Procedure GetString(Var buf:String) – reading of string
- Procedure GetNumber(Var num:Lfloat) – reading of number
- Procedure NextToken – reading of the next token
- Destructor Done – destructor

Unit L_Stream.pas

This unit declares an abstract stream interface – tStream and its successor – tFileStream(file access stream).

Global data types:

- TFStreamType = (FSTREAM_RESET,FSTREAM_REWRITE) – stream access type (read or write).

The description of tStream class.

Methods:

- Constructor Init – constructor
- Procedure BlockRead(Var Buf;Count:Longint) – read 'Count' bytes from the stream into memory block 'Buf'
- Procedure BlockRead(Var Buf;Count:Longint;Var ActualyRead:Longint) – read 'Count' bytes from the stream into memory block 'Buf' returning the number of bytes actually read
- Procedure Seek(I:Longint);Virtual;Abstract – set the read/write position
- Procedure ReadLine(Var S:String);Virtual;Abstract – read one text line
- Procedure WriteLine(S:String);Virtual;Abstract – write one text line
- Procedure BlockWrite(Var Buf;Count:Longint);Virtual;Abstract – write Count bytes from the memory block 'Buf' into the stream
- Procedure REW(I:Longint);Virtual – return I bytes into the stream
- Procedure FFWD(I:Longint);Virtual – skip I bytes in the stream

- Function GetPosition:Longint;Virtual;Abstract – current stream position
- Function GetSize:Longint;Virtual;Abstract – stream size of -1 if the size is undetermined
- Function EOF:Boolean;Virtual;Abstract – eod of stream flag
- Destructor Done – destructor

The description of tFileStream class.

Class tFileStream has the same list of procedures and an additional constructor InitFromFile(FileName:String) which opens FileName for reading or writing.

Unit L Stream.pas

This unit declares an abstract stream interface – tStream and its successor – tFileStream(file access stream).

Global data types:

- TFStreamType = (FSTREAM_RESET,FSTREAM_REWRITE) – stream access type (read or write).

The description of tStream class.

Methods:

- Constructor Init – constructor
- Procedure BlockRead(Var Buf;Count:Longint) – read 'Count' bytes from the stream into memory block 'Buf'
- Procedure BlockRead(Var Buf;Count:Longint;Var ActualyRead:Longint) – read 'Count' bytes from the stream into memory block 'Buf' returning the number of bytes actually read
- Procedure Seek(I:Longint);Virtual;Abstract – set the read/write position
- Procedure ReadLine(Var S:String);Virtual;Abstract – read one text line
- Procedure WriteLine(S:String);Virtual;Abstract – write one text line
- Procedure BlockWrite(Var Buf;Count:Longint);Virtual;Abstract – write Count bytes from the memory block 'Buf' into the stream
- Procedure REW(I:Longint);Virtual – return I bytes into the stream
- Procedure FFWD(I:Longint);Virtual – skip I bytes in the stream
- Function GetPosition:Longint;Virtual;Abstract – current stream position
- Function GetSize:Longint;Virtual;Abstract – stream size of -1 if the size is undetermined
- Function EOF:Boolean;Virtual;Abstract – eod of stream flag
- Destructor Done – destructor

The description of tFileStream class.

Class tFileStream has the same list of procedures and an additional constructor InitFromFile(FileName:String) which opens FileName for reading or writing.

Unit Logger.pas

This unit contains functions for debug information output into the external file.

Global variables:

- LogFile:Text – output file handle.

Procedures:

- Procedure InitLogger(FName:String) – open the output file.
- Procedure CloseLogger – close the debug information output file
- Procedure PushProc(P:String) – trace a procedure call
- Procedure PopProc – trace a return from procedure
- Procedure Log(S:String) – write a text string into output file
- Procedure LogF(F:Single) – write a number into output file
- Procedure Fatal(S:String) – abnormal program termination

Unit L_Types.pas

This unit contains declarations of globally used data types.

Here is the list of defined types:

1. Basic scalar types

- Lfloat=Single;
- Ldouble=Double;
- Luint=LongWord;
- Lsizei=LongInt;
- Lclampf=Single;
- Lclampd=Double;
- Lbyte=ShortInt;
- Lubyte=Byte;
- Lenum=Cardinal;
- Lboolean=Boolean;
- Lbitfield=Cardinal;
- Lshort=SmallInt;
- Lint=Longint;
- Lushort=Word;
- PLvoid=Pointer;
- PLenum=^Lenum;
- PLsizei=^Lsizei;
- PLboolean=^Lboolean;
- PLbyte=^Lbyte;
- PLshort=^Lshort;
- PLint=^Lint;
- PLubyte=^Lubyte;
- PLushort=^Lushort;

- PLuint=^Luint;
- PLfloat=^Lfloat;
- PLclampf=^Lclampf;
- PLdouble=^Ldouble;

2. Structured types

- pMatrix3 = ^tMatrix3 – a pointer to 3x3 matrix
- pMatrix4 = ^tMatrix4 – a pointer to 4x4 matrix
- tMatrix3 = Array[1..3,1..3] Of Lfloat – 3x3 matrix
- tMatrix4 = Array[1..4,1..4] Of Lfloat – 4x4 matrix
- pVector3=^tVector3 – a pointer to vector
- tVector3=Array[1..3] Of Lfloat – a vector in the array form
- pVectorXYZ = ^tVectorXYZ – a pointer to 3D vector
- tVectorXYZ = Record X,Y,Z:Lfloat;End – a 3D vector

Unit Man_Obj.pas

This unit implements tManipulatorModel class which render the virtual manipulator model.

The description of tManipulatorModel class

Methods:

- Constructor Init – constructor
- Procedure CreateLists – creation of some internal OpenGL geometry lists
- Procedure Render – rendering of virtual manipulator model
- Function GetJointAngle(Index:LongInt):Lfloat – get joint angle
- Procedure SetJointAngle(Index:LongInt;Val:Lfloat) – set joint angle
- Procedure SelectJoint(JIndex:LongInt) – joint selection
- Function GetJoint:LongInt – get current joint index
- Function GetCurrentPoint(angles:array of GLDouble;var joint6Point:TPoint):Tpoint – coordinate of the last manipulator link
- Destructor Done - destructor

Unit StatusW.pas

This unit implements tStatusWindowManager class which manages the debug information output window.

The description of tStatusWindowManager class:

Fields:

- StatusHWnd:HWnd – window handle

Methods:

- Constructor Init – constructor

- Procedure CreateStatusWindow – window creation
- Procedure RenderStatus(dc:HDC) – debug information output
- Procedure Update – update window contents
- Procedure RegisterStatusWindowClass – window class registration
- Destructor Done – destructor

Unit Vars.pas

This unit implements internal variables management system.

The description of tVarsManager class:

Fields:

- NumVars:LongInt – number of reistered variables
- Variables:Array Of Record Name:String;ValueS:String;ValueF:Lfloat;IsExternal:Boolean;ExternalType:tVarType;ExternalPtr:Pointer; End – the list of variables

Methods

- Constructor Init – constructor which registers some basic variables from Consts.pas unit
- Function FindVariable(Name:String):LongInt – variable search in the list
- Procedure SetVariableC(Params:String) – set variable value
- Procedure SetVariablefC(Params:String) – set numeric variable value
- Procedure MultiplyByC(Params:String) – multiply variables
- Procedure DivideByC(Params:String) – divide variable 1 by variable 2
- Procedure AddToC(Params:String) – varaibles addition
- Procedure SubFromC(Params:String) - variables substraction
- Procedure PrintVarC(Params:String) – output variable value into file
- Procedure AddVariable(Name:String;Value:String) – add string variable
- Procedure AddVariable(Name:String;Value:Lfloat) – add numeric variable
- Procedure AddExternalVariable(Name:String;PtrTo:Pointer;VType:tVarType) – add external (not declared in the program) variable
- Procedure SetVariableValue(Name:String;Value:Lfloat) – set numeric variable value
- Procedure SetVariableValue(Name:String;Value:String) – set string variable value
- Function GetFloat(Name:String):Lfloat – get numeric variable value
- Function GetString(Name:String):String – get string variable value
- Destructor Done – destructor

Unit B_Bitmap.pas

This unit declares the tBitmap class which implements loading and saving of Windows Bitmap files (.BMP).

Global data types:

- tRGB = Record R,G,B:Byte ; End - R/G/B (red/green/blue) triplet
- pRGB = ^tRGB - a pointer to tRGB (used for array addressing)

tBitmap class description.

Fields:

- Width:LongInt – image width in pixels
- Height:LongInt – image height in pixels
- Bits:LongInt - color depth in bits(only 32 bit images are supported)
- Data32:pRGB – pointer to 32 bit image data

Methods:

- Constructor Init(W,H:LongInt) – constructor which creates an empty image of the specified size
- Constructor InitFromFile(FileName:String) – constructor which loads an image from external file 'FileName'
- Procedure LoadBMP(FName:String) – procedure for loading image from file.
- Procedure SaveBMP(FName:String) – procedure for saving image to file
- Procedure FreeBitmap – frees memory allocated for image
- Procedure ReallocBitmap – image memory allocation
- Destructor Done – destructor which frees the memory

Unit V_VidStr.pas

This unit declares an abstract interface for video capture.

Methods of tVideoStream:

- Constructor Init – constructor
- Procedure ResetStream;Virtual;Abstract – start video captue
- Function GetNextFrame:pBitmap;Virtual;Abstract – get next video frame
- Function GetNextFrameM(ptr:pByte):integer;Virtual;Abstract – get next frame with possible error code
- Destructor Done;Virtual – destructor

Unit Utils.pas

This unit implements miscelaneous data conversion routines.

Functions:

- Function Int2Str(I:Longint):String – integer to string conversion
- Function FLT2Str(I:Lfloat):String – floating point number to string conversion
- Function FLT2Str(I:Lfloat;N:Longint):String – floating point number to string conversion with exact number of decimal digits

- Function GLD2Str(I:Ldouble):String – double precision number to string conversion
- Function GLD2Str(I:Ldouble;N:Longint):String – double precision number to string conversion (the same as FLT2Str)
- Function Str2Int(I:String):Longint – string to integer conversion
- Function HexStr2Int(Str:String):LongInt – conversion of string with hexademical number into long integer
- Function Str2GLD(I:String):Ldouble – string to double precision number conversion
- Function Str2FLT(I:String):Lfloat – string to floating point conversion
- Function IsCorrectInt(I:String):Boolean – check string validity (if it is an integer)
- Function IsCorrectGLD(I:String):Boolean – check string validity (if if is a floating point number)
- Function Bool2Str(I:Boolean):String – conversion of boolean value into the string
- Function Str2Bool(I:String):Boolean – conversion of string into boolean value
- Function UpCaseStr(S:String):String – upper case conversion
- Function BinStr(Val:Longint;Cnt:Byte):String – binary integer into string conversion
- Function HexStr(Val:Longint;Cnt:Byte):String – hexademical number into string conversion
- Function GetToken(S:String;Num:Byte):String – Nth token from string S

Unit Timing.pas

This unit implements tCPUTimer class which is used for precise timing

The description of tCPUTimer class:

Fields :

- NumOfProcessors:LongInt – number of processors
- TimingList:pTimingList – the list of timing sections
- CyclesPerSecond:LongInt – CPU clock rate

Methods :

- Constructor Init – initiate timing system
- Function GetTime:Int64 – current system time
- Procedure PushTimingSection(Name:String) – start new timing section
- Function GetSectionRunningTime:Int64 – current sectino running time
- Procedure PopTimingSection – stop current timing section
- Function GetCyclesPerSecond:LongInt – CPU clock rate
- Function GetTimingString:pLongStr – the list of embedded timing sections
- Destructor Done – destructor

Unit Win32Aux.pas

This unit implements auxillary functions for main window initialisation, destruction and event handling.

Global variables:

- _Hwnd:hWnd – main window handle
- Dc:Hdc – main window's device context handle
- Rc:Hglrc – OpenGL device context handle

Procedures:

- Function CreateGLWindow(Title:String;Width,Height,Bits:Longint;StereoMode:Boolean):Boolean – main window creation using Title,Width and Height parameters

- Procedure KillGLWindow – destruction of main application window and associated device contexts. This function is called before program termination

- Function glWindowProc() – standart event handling function. It is called implicitly in main.pas.

Here is the list of events handled by glWindowProc() function :

- WM_LButtonDown,WM_RButtonDown,WM_MbuttonDown/WM_LButtonUp,WM_RButtonUp,WM_MButtonUp – mouse button press/release events.KeyMapper and EventMAnager are called.
- WM_KeyDown,WM_SysKeyDown/WM_KeyUp,WM_SysKeyUp – key press/release. KeyMapper and EventManager are called
- WM_Activate/WM_Deactivate – window activation/deactivation event. The methods EventManager.Activate and EventManager.Deactivate are called
- WM_Close – close the window. Program execution stops

The relation diagram of classes used the first stream is shown in Fig.4.12.

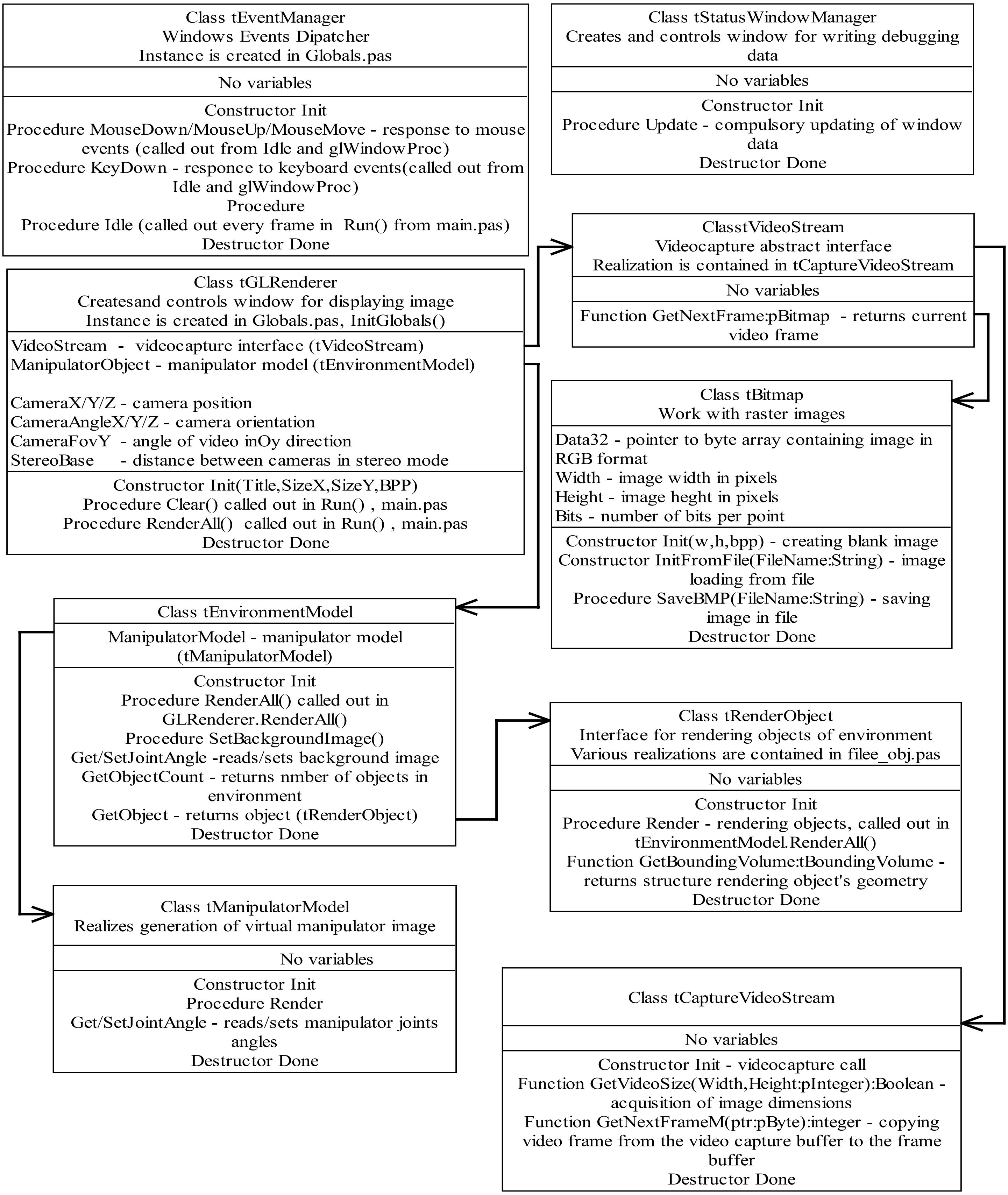


Fig. 4.12. Relation diagram of used classes

4.2.4.1.3 Block diagrams of the main program and procedures and functions it uses are given below for a detailed description of the graphic stations' SW.

Block diagram of main program (it's text is in the file Main.pas) is shown in Fig. 4.13.

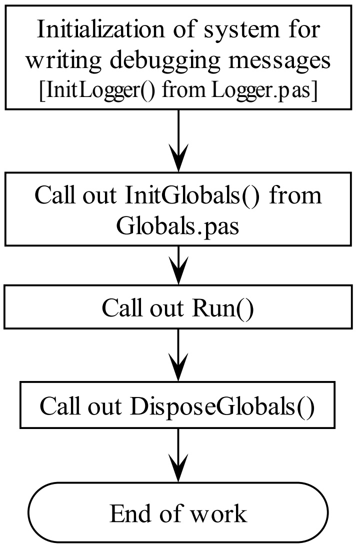


Fig. 4.13. Block diagram of the main program.

Within the main program functions for initialization of global variables (InitGlobals()) are called out and the cycle for handling events is run (Run()). In the end of operation a procedure for destruction of all created objects is called (DisposeGlobals()).

A block diagram of the InitGlobals() procedure, mentioned in the main program's block diagram Global.pas, is shown in Fig.4.14. Its text is in a file Global.pas.

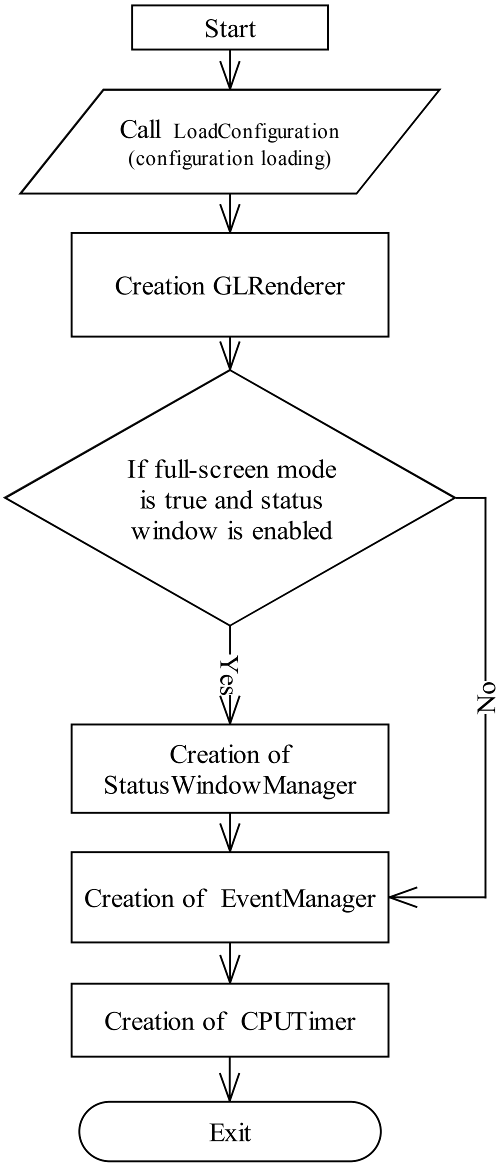


Fig.4.14. A block diagram of InitGlobals() procedure.

InitGlobals() procedure creates all objects used in the program and initializes all global variables declared in Consts.pas.

A block diagram of Run() procedure, mentioned in the main program's block diagram, is shown in the Fig.4.15.

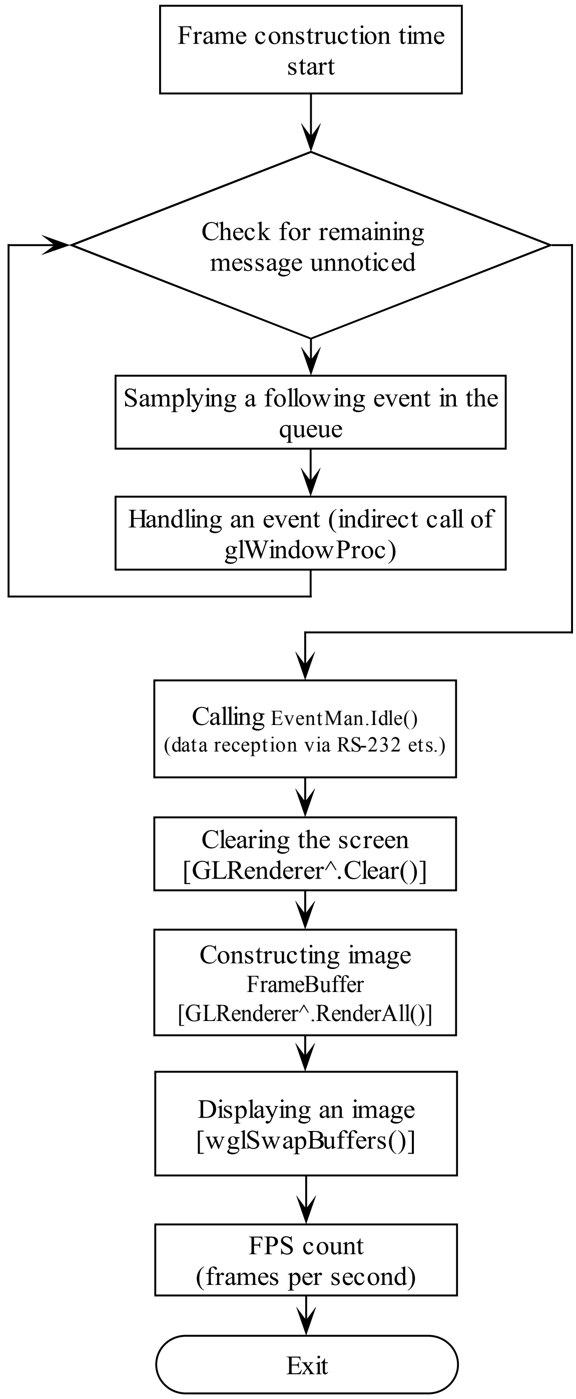


Fig.4.15. A block diagram of Run() procedure.

The Run() procedure realizes the cycle of handling events coming from external devices and constructing the augmented image. Here objects EventMan and GLRenderer are used.

A function glWindowProc(), a part of Run() procedure, is a standard windowing function of Windows (Fig.4.16) and its text is in Win32Aux.pas.

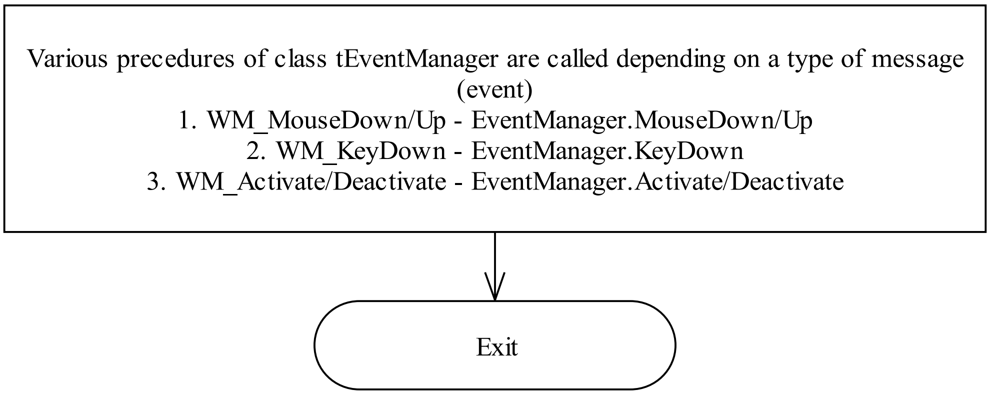


Fig.4.16. A block diagram of glWindow Proc()

This function is needed for Window control in Windows system

A block diagram of DisposeGlobals(), mentioned in the main program's block diagram, is shown in the Fig. 4.17.

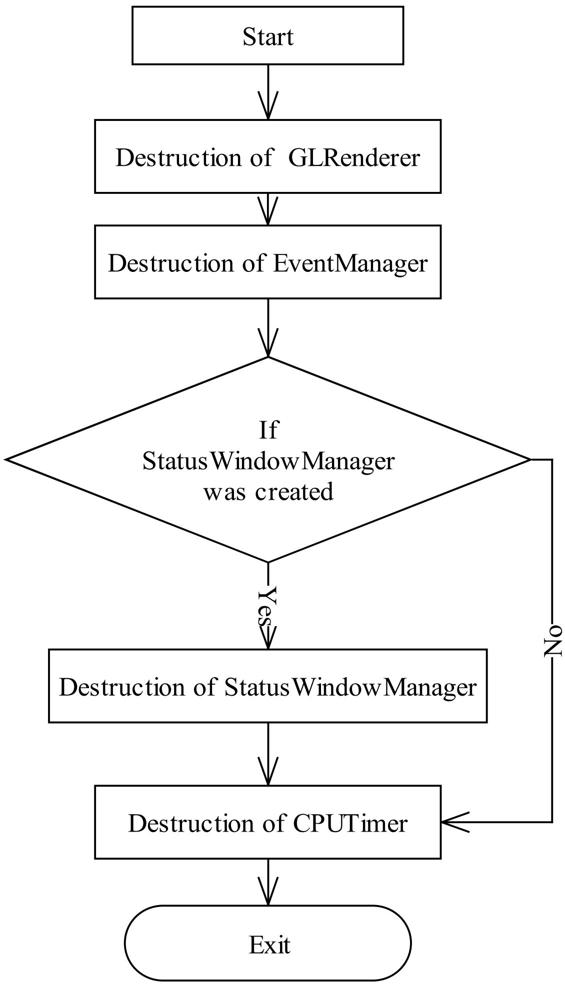


Fig.4.17. A block diagram of DisposeGlobals() procedure.

DisposeGlobals() procedure destructs all created objects.

A block diagram of tEventManager.Idle() procedure, a part of RVP() procedure, which text is in a file EventMan.pas, is shown in Fig.4.18.

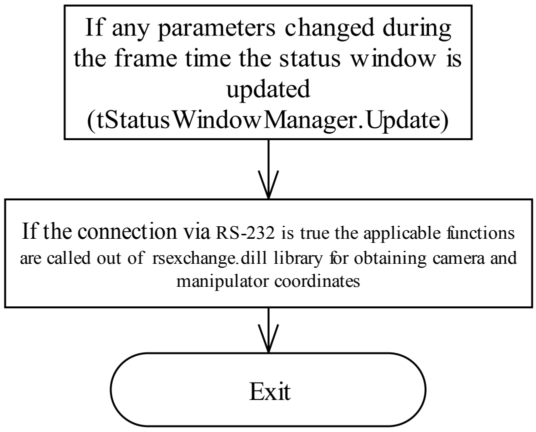


Fig.4.18. A block diagram of tEventManager.Idle().

A procedure tEventManager.Idle() is called each frame of Augmented Reality. It updates the window of debugging messages and receives via COM-port (RS-232) position/orientation coordinates of the environment observation camera. A dynamically loaded library rsexchagne.dll is used for that, whose description is given below.

A block diagram of tGLRenderer.RenderAll procedure, a part of RUN() procedure, whose text is in a file R_Render.pas is shown in Fig.4.19.

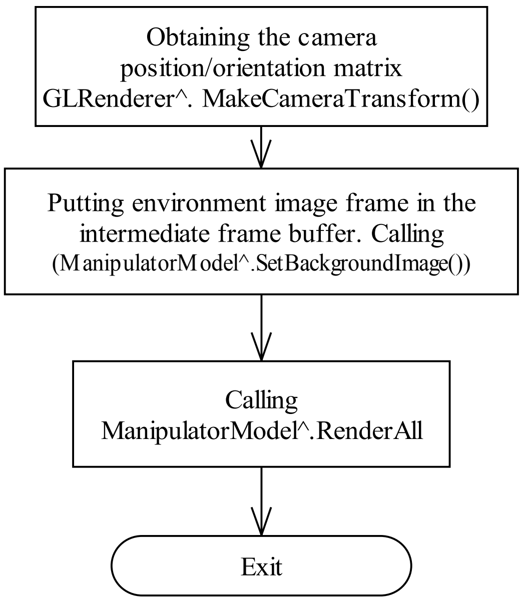


Fig.4.19. A block-scheme of tGLRenderer.RenderAll() procedure.

A tGLRenderer.RenderAll() procedure prepares input data and constructs an environment image with them augmented with the virtual manipulator's image (Augmented Reality).

A block diagram of tGLRenderer.MakeCameraTransform procedure, a part of the diagram in Fig. 4.19, is shown in Fig.4.20.

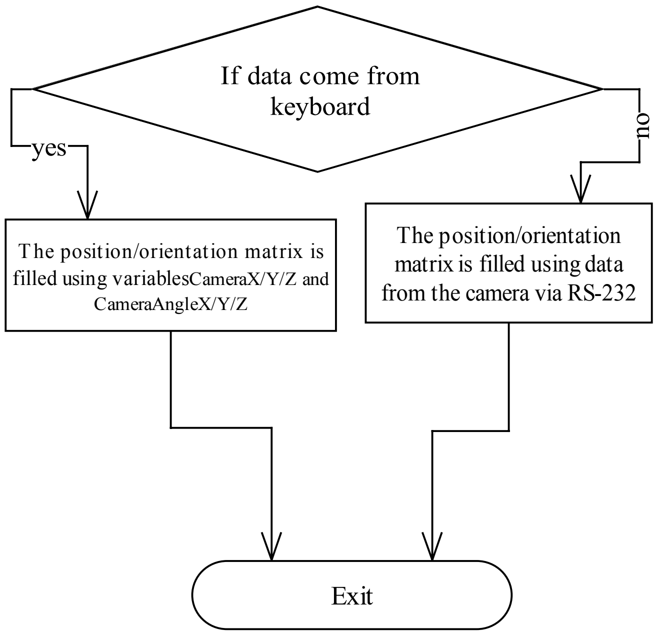


Fig.4.20. A block diagram of tGLRenderer.MakeCameraTransform procedure.

A tGLRenderer.MakeCameraTransform() procedure fills the camera position/orientation matrix for obtaining the augmented image of environment.

A block diagram of a tEnvironmentModel.RenderAll() procedures class is shown in Fig.4.21 (texts of procedures are in a file E_Man.pas). A procedure ManipulatorModel^ (Fig.4.19) belongs to this class, it directly realizes the augmented reality imaging.

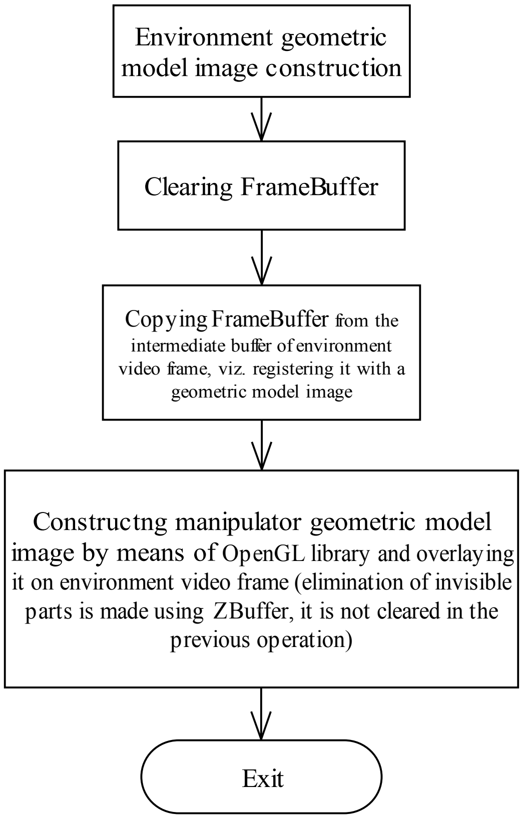


Fig. 4.21. A block diagram of tEnvironmentModel.RenderAll() procedure.

<u>Dynamically loaded library RsExchange.dll</u> is purposed for reception of position/orientation coordinates of real environment observation camera from Unit 2 («SPHERA-36»).

A language used for realization is C^{++}, a compilator is gcc (GNU C).

This library exports the following functions:

int _stdcall InitSferaExchange(char*port);

int _stdcall SetSferaData(DataToSfera*data);

int _stdcall GetSferaData(DataFromSfera*data);

Classes used in the library are: SferaExchnage, SferaThread.

The following files contain the initial texts:

Makefile - file needed for compiling the library with the help of utility GNU make;

RSExchange.cc - file with texts of exported functions;

RSExchange.def - file with names of exported functions;

RSExchange.h - header file with description of exported functions;

SferaExchange.cc - file with texts of functions SferaExchange class;

SferaExchange.h - header file with description of SferaExchange class;

SferaThread.cc - file with texts of SferaThread class functions;

SferaThread.h - header file with description of SferaThread class;

SferaTypes.h - header file with description of data types used in the library.

The exported functions execute the following operations:

int __stdcall InitSferaExchange(char*port);

The function opens a COM-port with a name 'port' and creates a stream that begin to receive data from «Sphera-36» by way of creating a SferaThread class object and calling start functions of this class.

int __stdcall SetSferaData(DataToSfera*data);

Puts data from the memory area at pointer 'data' to the buffer for transmitting to «SPHERA-36» calls sendDataToSfera function of SferaThread class.

int __stdcall GetSferaData(DataFromSfera*data);

Puts data from «Sphera-36» to the area of pointer 'data' (calls getDataFromSfera function of SferaThread class).

A class SferaExchange is purposed for data exchange with «Sphera-36».

The class contains the following functions in a section “public”:

```
SferaExchange(string port);
int receiveData();
int sendData();
void getData(DataFromSfera*data);
void setData(DataToSfera*data);
```

Also, in a section “protected” there are the following variables:

```
DataFromSfera dataFromSfera; // buffer for receiving data
DataToSfera dataToSfera; // buffer for transmitting to «Sphera-36»
```

A constructor of a class SferaExchange(string port) opens the COM-port with a name 'port' and sets its parameters:

speed - 9600 bits/s;

paruty bit - no;

stopping bits - 2.

A function int receiveData() is purposed for receiving data from Unit 2 (SPHERA-36). It provides reception of 3x3 camera rotation matrix and 3 camera coordinates. All data are put in the receiving buffer - dataFromSfera. The function waits for coming data during 50 ms and, if data are none, the receiving buffer remains unchanged and the function is terminated.

A function int sendData() is purposed for transmitting data to Unit 2 («SPHERA-36»). Data are sent from a buffer data dataToSfera.

A function void getData(DataFromSfera*data) serves for transmitting received data from a buffer data dataFromSfera to a user program. A content of the receiving buffer is copied to a memory area at a pointer 'data'.

A function void setData(DataToSfera*data) from a memory area at a pointer 'data' to a buffer dataToSfera for transmitting to «SPHERA-36».

A class SferaThread is purposed for creating a stream wherein the data exchange with «SPHERA-36» is realized. The data exchange uses a class SferaExchange.

A class has the following functions in a section 'public':

```
SferaThread(string sferaCom);
~SferaThread();
void sferaRun();
void start();
void sendDataToSfera(DataToSfera*dataToSfera);
void getDataFromSfera(DataFromSfera*dataFromSfera);
```

The constructor of SferaThread(string sferaCom) class creates objects of SferaExchange class and the data exchange stream.

The destructor of ~SferaThread() class cancels the data exchange stream.

A function void sferaRun() is run as a stream: a function receiveData of SferaExchange class is called in repeated cycle waiting for data to put them in a reception buffer of data dataFromSfera of SferaExchange class.

A function void start() runs the data exchange stream.

A function void sendDataToSfera(DataToSfera*dataToSfera) puts data from a memory area at a pointer 'dataToSfera' in the buffer for transmitting to «SPHERA-36» (calls a function setData of SferaExchange class).

A function void getDataFromSfera(DataFromSfera*dataFromSfera) copies content of the reception buffer to a memory area at a pointer 'dataFromSfera' (calls a function getData of SferaExchange class).

A given below block diagram (Fig.4.22) illustrates the stream of data received from Unit 2 («SPHERA-36»).

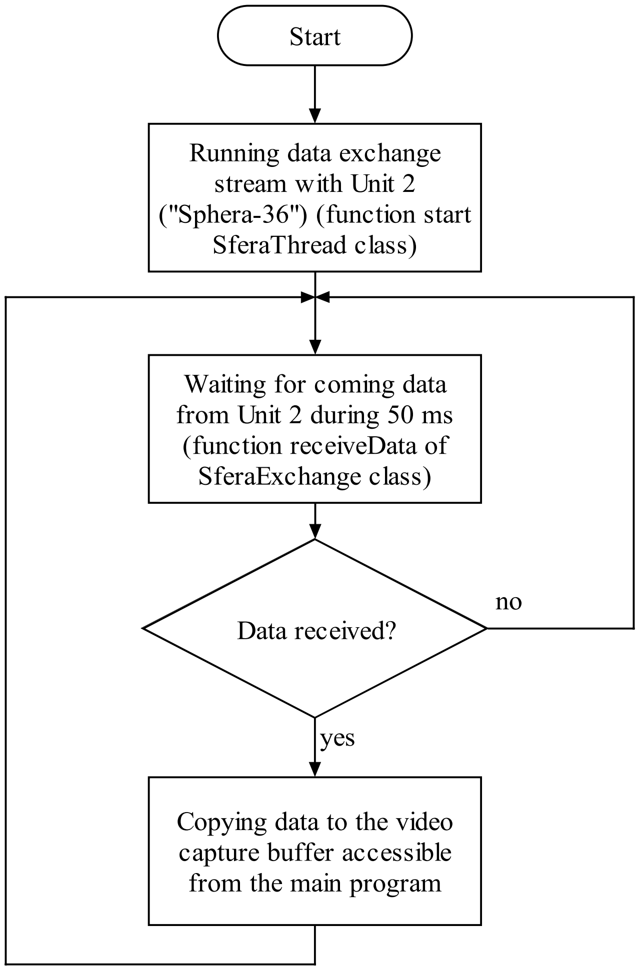


Fig.4.22. Block diagram illustrating a steam for receiving data from Unit 2 («SPHERA-36»).

A dynamic library dshowtest.dll is purposed for the video frame capture using a program interface (API) DirectShow of Microsoft DirectX.

The realization language is C++, the compilator is Microsoft Visual C++ 6.0.

This library exports the following functions:

```
_declspec(dllexport) int InitCapture(HWND parentWin,int width,int height);
_declspec(dllexport) int GetImage(char*prt);
_declspec(dllexport) int GetImageSize(int*width,int*height);.
```

It also uses auxiliary functions:

```
HRESULT GetInterfaces(void);
HRESULT FindCaptureDevice(IBaseFilter ** ppSrcFilter);
HRESULT CaptureVideo(HWND pwin,int width,int height,MyRenderer**);
void Msg(TCHAR *szFormat, ...);
```

and a class MyRenderer for processing captured frames.

Texts of the functions are kept in the following files:

MyRenderer.cpp - file with texts of functions of MyRenderer;
MyRenderer.h - file with description of SferaExchange;
dshowfunc.cpp - file with texts of auxiliary functions;
dshowfunc.h - file with description of auxiliary functions;
dshowtest.cpp - file with texts of exported functions.

The exported functions execute the following operations:

A function_declspec(dllexport) int InitCapture(HWND parentWin,int width,int height) initializes and triggers the video capture with resolution 'width' x 'height' by way of calling a function CaptureVideo.

A function _declspec(dllexport) int GetImage(char*prt) puts the latest captured video image frame to a memory area at a pointer 'ptr' with the help of a function getImage of MyRenderer class.

A function _declspec(dllexport) int GetImageSize(int*width,int*height) is purposed for setting resolution of a captured video images: in a variable at a pointer 'width' an image width is placed, at a pointer 'height' - its height for what a function getImageSize of MyRenderer class is called.

An auxiliary function HRESULT GetInterfaces(void) obtains the pointers to structures and classes necessary for the video capture control.

An auxiliary function HRESULT FindCaptureDevice(IBaseFilter ** ppSrcFilter) serves for searching in the video capture operation system.

An auxiliary function HRESULT CaptureVideo(HWND pwin,int width,int height,MyRenderer**) initializes video capture using functions GetInterfaces and FindCaptureDevice and upon that triggers capture.

An auxiliary function void Msg(TCHAR *szFormat, ...) is purposed for displaying error messages at capture initialization.

A class MyRenderer:public CBaseVideoRenderer is purposed for video capture provision by copying video frames from the system memory to a buffer allocated in the library. The class has the following functions in a "public" section:

```
MyRenderer(LPUNKNOWN pUnk,HRESULT *phr);
~MyRenderer();
HRESULT CheckMediaType(const CMediaType *pmt );
HRESULT SetMediaType(const CMediaType *pmt );
HRESULT DoRenderSample(IMediaSample *pMediaSample);
void getImageSize(int*width,int*height);
void getImage(char*ptr);
```

<u>A constructor of a MyRenderer(LPUNKNOWN pUnk,HRESULT *phr) class calls a constructor of the parent class.</u>

A destructor of ~MyRenderer() clears the memory allocated for the video frame buffer.

A function HRESULT CheckMediaType(const CMediaType *pmt) is called by DirectX for checking admissibility of a set capture format.

A function HRESULT SetMediaType(const CMediaType *pmt) is called by DirectX at setting a video format. It allocates the frame buffer memory of a size needed for a current video frame resolution.

A function HRESULT DoRenderSample(IMediaSample *pMediaSample) is called upon a frame capture. It copies the frame in a buffer allocated in a function SetMediaType.

A function void getImageSize(int*width,int*height) is purposed for obtaining a current video frame capture resolution. A frame width and height are put in variables at pointer 'width' and 'height', respectively.

A function void getImage(char*ptr) is purposed for a user program's obtaining the frame buffer content. The buffer allocated in a SetMediaType function is copied to a memory area at a pointer "ptr".

4.2.4.2 The algorithm and SW prototype for generation data needed for force interaction

The algorithm ofdata generation needed for realizing force interaction of virtual manipulator's with environmental objects was described in detail in par.1.1 of Report # 3 on Task 6 and par.3 of this report.

This algorithm provides:

- reception from the force interaction system of unit 9 realized by SPHERA-36' central processor of the master arm's joint generalized coordinates vector value that is the desired value of the virtual manipulator's generalized coordinates vector g_d, on condition that collisions are none, and, also, reception to this vector of correction vector value Δg that is nonzero value whenever the virtual manipulator has collisions with environment;
- detection of events of the virtual manipulator's impacting with environmental objects;
- determination of parameters of obstacles the virtual manipulator impacted with, precisely, the unity outward normal q_e in an collision point and coordinates of this point of collision x_c; these coordinates are needed for the SPHERA-36 computer for computing an expected force of virtual manipulator's interaction with environment and a value of joint coordinates correction vector Δg;
- transmission of the computed values q_e and x_c, and, also, status of the collision flag to SPHERA-36' computer.

The realized version of the subsystem prototype for the virtual manipulator's interacting with environment arm is designed not for three, as was earlier planned (see Report # 3, Task 6), but for one possible type of work tool's collision with environmental objects precisely, for collision of vertex of polyhedron approximating the tool with a surface (the first type of interaction according to the classification given in part 1.2 Report # 3 on Task 6). Moreover, this vertex is one and is the tool's tip lying on the axis of symmetry.

This simplifies realization of the interaction subsystem and, in the same time, does not bar ascertaining acceptability of the proposed principles for building the interaction subsystem and of the method for implementation of such subsystems.

The algorithm's block diagram is shown in Fig. 4.23.

The algorithm operates cyclically. The first operation of the k^{th} cycle of the algorithm is ascertaining the presence of the sync byte that is sent to the graphic station every 64 ms by the module for generating the vector of forces of the virtual manipulator's interaction with environment operating on the base of SPHERA-36' central processor.

If the sync byte is none the control is returned to the basic program of the graphic station executing the functions of modeling and displaying the manipulator and its environment (scene). If the sync byte is present the following data, computed on the previous step (k-1), are transmitted to the program module of the SPHERA-36' central processor from the output data array locOutData:array[0…30]: status of the flag$^{(k\text{-}1)}$ indicating collision with environment, the vector of contact point $x_c^{(k-1)}$ and the unity outward normal $q_c^{(k-1)}$ to the object's surface in this point (block 2 of the algorithm, see Fig.4.23).

Then there come from the SPHERA-36'central processor to the input data locInData:array[0…30] the joint (generalized) coordinates current vector for the master arm $g_d^{(k)}$ pertaining to the k^{th} step, and the correcting value $\Delta g^{(k)}$ for this vector (block 3) and the received data are copied in the buffer InBuf:array[0…30] (block 4).

Then there go obtaining the vector of generalized coordinates from the buffer InBuf:array[0…30] (block 5) and ascertaining identity of vector $g^{(k)}= g_d^{(k)} +\Delta g^{(k)}$ to vector $g^{(k\text{-}1)}$ of joint coordinates for the previous $(k\text{-}1)^{\text{th}}$ step (block 6). And if the identity really is, the variables flag $^{(k\text{-}1)}$, $x_c^{(k)}$, $q_c^{(k)}$, that are to be transmitted to the SPHERA-36' central processor, are given the same values as were on the previous step and the control is transferred to the basic program of the Graphic Station. If not, i.e. $g^{(k)} \neq g^{(k\text{-}1)}$, position vectors $x^{n\text{-}1}$ and x^n are computed, of the origin and the tip of the virtual manipulator's last link (block 7), it is obvious that this tip is the point of possible contact with environment.

The next algorithm's block (8) performs determination of intersection points $x_i^1, x_i^2, ..., x_i^n$ with surface of environmental objects of the line going from point x^{n-1} to point x^n.

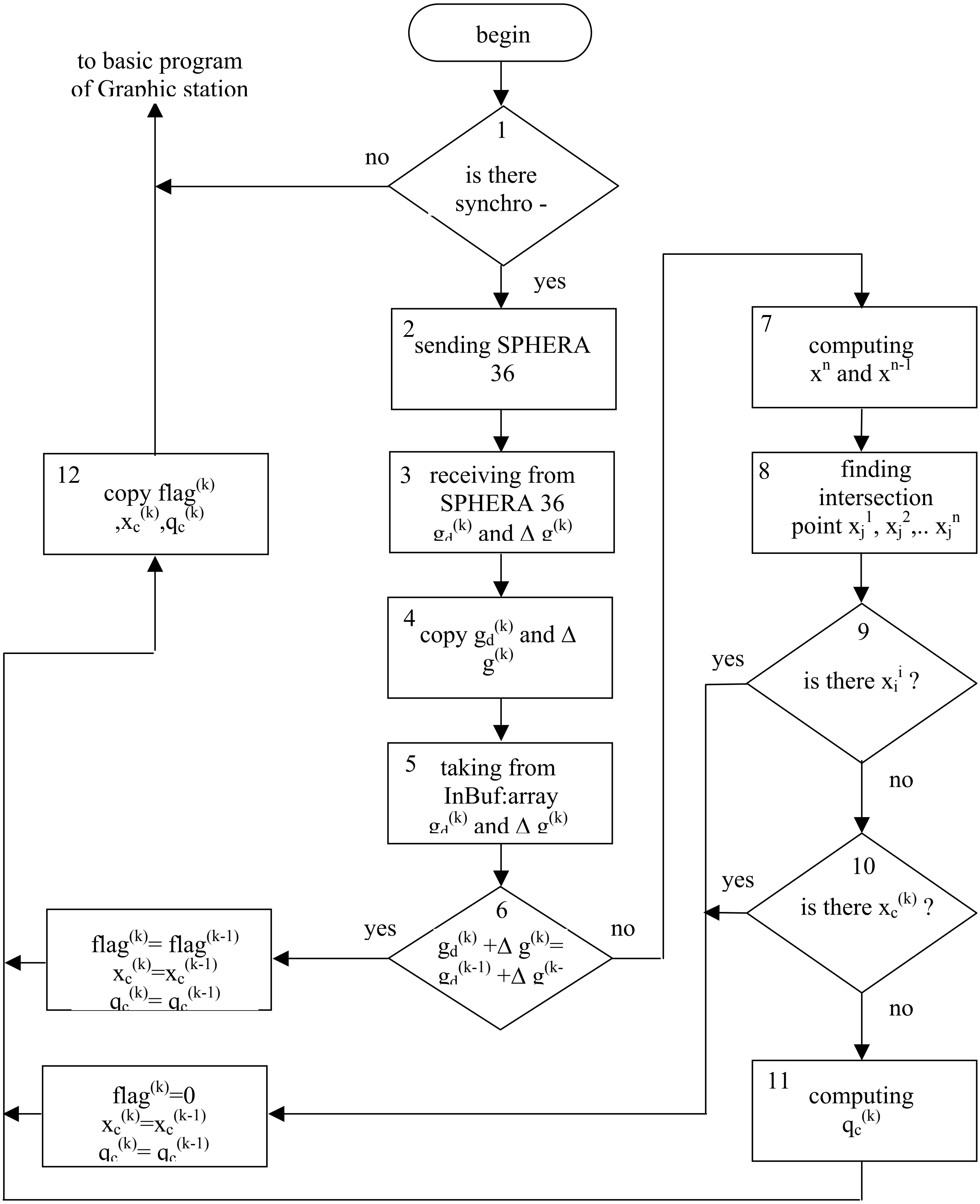


Fig. 4.23 Block-diagram of algorithm for detection of collision of virtual manipulator and environment

In this variant of the reactant interaction system implementation it was assumed that all environmental objects are cylinders or truncated cones (Fig.4.24) and the line intersection points are computed as lying on surfaces of these bodies.

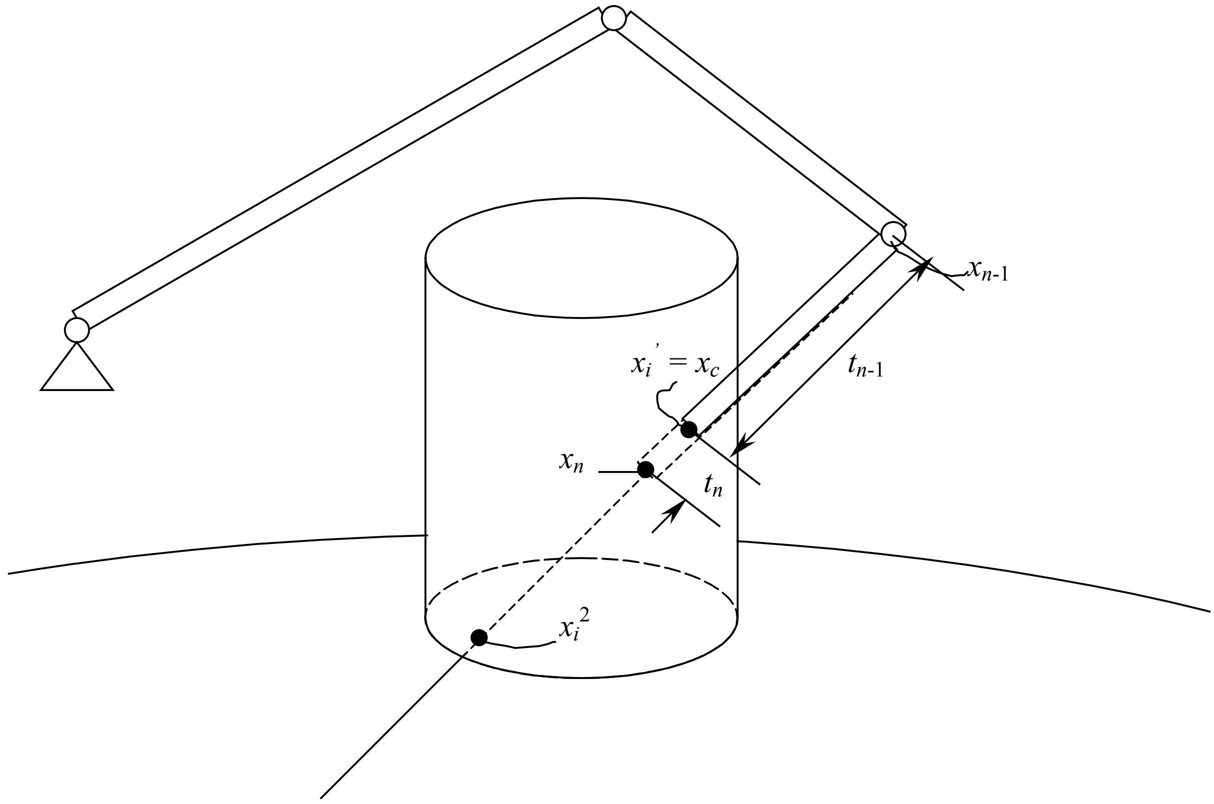


Fig. 4.24 Illustration to definition of possible intersection points
of the manipulator with environment

Surely, the intersection points may not exist what is verified with the algorithm's 9 block. In this case the contact flag$^{(k)}$ is considered cleared and the rest data transmitted to SPHERA-36' central processor do not change their values and after that control is passed to the basic program of the Graphic Station. Note, that if a condition flag$^{(k)}$=0 is satisfied the algorithm does not compute the values characterizing manipulator's collision with environment: G_d vector $G^{(k)}$ and correction vector to the vector $g^{(k)}$ of generalized coordinates. Therefore, by flag$^{(k)}$=0 values $q_c^{(k)}$ and $x_c^{(k)}$ may be any, e.g. zero for definiteness. If flag$^{(k)}$=1, that is intersection points exist, even then the point of contact may not be found.

In order to find this contact point out of set $x_i^1, x_i^2, ..., x_i^n$ it is needed that vector x_i^i corresponding to it would satisfy a condition below. The algorithm's block 10 tries satisfaction of these conditions.

$$t_{n-1} + t_n \leq l_n + \Delta ,$$

where t_{n-1}, t_n - distance of intersection point x to the origin and the end of the n^{th} link, respectively;

l_n - length of n^{th} link;

Δ - tolerable error.

If no one point satisfies this condition control is passed to the algorithm's branch of the previous case when intersection points are absent.

If a contact point exists, i.e. the above condition is satisfied for one point of the set of intersection points the appropriate buffer locOutData:array[0…30] are written the indication of contact (flag$^{(k)}$=1) and coordinates of the found contact point $x_c^{(k)}$ for subsequent transmitting them to SPHERA-36' central processor. Then the unity normal to the surface in the contact point $q_c^{(k)}$ is computed with subsequent writing to the buffer for transmitting to SPHERA-36. After that control is passed to the basic program of the Graphic Station.

The developed algorithm is programmed in Delphi. It is composed of two parts.

The first part computes the input data for the program installed in the SPHERA-36'central processor generating forces of interaction. The second part provides the data exchange via communication channel RS-232 between the just mentioned program and the considered program installed in the Graphic Station.

The first part is an addition to the first stream of Graphic Station SW which provides generation of 3 D images of the real environment augmented by the virtual manipulator for their known geometric models and observer's position.

The developed addition generates the contact flag and computes the contact point vector x_c and the unity normal to the surface in this point q_c for the current values of the master arm vector g_d of generalized coordinates.

This part is described in modules UnitLook.pas and Normal.pas. It should be noted that in UnitLook the part of Manipul program also described, one generating 3D images.

The second part is the data exchange stream via communication channel RS-232. Input data coming from the force interaction system of Unit 9 - SPHERA-32' central processor are current values of vector of master arm's generalized coordinates g_d and the correction vector Δg. The output ones are flag, x_c , q_c coming from unit 7 (Graphic Station 1).

This part is described in module ComThread.pas.

This modules are presented below. Note, that module UnitLook.pas is given not full. Only variables, procedures and functions are given, ones used by the first part of the developed software prototype. The rest part of this module is the basic software of the Graphic Station realizing generation of 3D images of the virtual manipulator and its environment.

UnitLook.pas module

Constants:

NumbersOfCylinders – number of environmental objects;

cylinders:array[0..NumberOfCylinders-1]of Tcylinder – description of objects.

Global variables:

Angle_Joint:array [0..6]of double – virtual manipulator generalized coordinates data array used for visualization.

Link_Lenght:array [0..6]of TLenght – virtual manipulator link length array used for visualization and computation of point of contact with environment.

FrmLook:TFrmLook – object of a class type of window in virtual manipulator's representation.

Variables of TFrmLook class

private

InBuf:array[0..30]of SmallInt – array of data transmitted from the SPHERA-36'central processor to the Graphic Station via RS-232.

outBuf:array[0..30]of SmallInt – array of data transmitted from the Graphic Station to the SPHERA-36'central processor via RS-232.

colFlag: byte – flag of virtual manipulator's contact with the work scene.

comTh: XcommThread – object of data exchange stream with robot.

public

RealJoints:array[0..5]of double – master arm actual generalized coordinates (without correction).

Functions and procedures of TFrmLook class

Procedure myOnIdle(Sender: TObject; var Done: Boolean) – called when no events take place in the system.

Function currentManipulPoint(angles: array of GLDouble; jointLens: array of TLenght; var joint6Point: TPoint): TPoint – computes 3D coordinates of the origin and the tip of the virtual robot's last link for the current values of generalized coordinates.

angles - generalized coordinates.

jointLens – virtual robot link lengths.

joint6Point – variable corresponding to coordinates of the virtual manipulator's last link.

ComThread.pas module

This module describes the data exchange stream with the real robot-manipulator via RS-232 interface – XCommThread.

Variables of XCommThread class

comHandle:THandle – descriptor of the device (handle) file (COMx)

inData:PWordArray – pointer to array in the main stream of received data

outData:PWordArray – pointer to array in the main stream of transmitted data

pCollisionFlag:PByte – pointer to flag of robot contact with surface in the main stream

locInData:array[0..30]of word – array of received data

locOutData:array[0..30]of word – array of transmitted data

collision Flag:byte – flag of robot's contact with environmental surface

Procedures XCommThread class

Constructor Create(port:PChar; inArray:PWordArray; outArray:PWordArray; pFlag:PByte) – stream constructor opening file linked with COM- port and setting transmission rate 9600 bit/s, byte size – 8 bit without even parity bit. Arguments: port – name of device file, inArray – pointer to buffer of data received from SPHERA-36, outArray – pointer to buffer of transmitted data.

Procedure Execute – organizes the cycle of transmitted data stream. It provides waiting for start byte from SPHERA-36'central processor (initiating the exchange) upon reception of which transmitted: flag of contact with surface, contact point coordinates and components of unity normal to the surface. Next, master handle generalized coordinates are received. On cycle completion the function CopyData is called.

Procedure CopyData – copies (writes) received data in buffer inData for the main stream and, also, copies data generated by the main stream to buffer outData for their subsequent passing to the data exchange stream from this buffer.

Normals.pas module

Functions

Function FindCylindersNormal(point, prevPoint: TPoint; cylinders: array of TCylinder; n:integer; var touchPoint, normal: TPoint):boolean – the function determines virtual robot's point of contact with environmental object surface, unity normal to the surface in the contact point and the contact flag. Arguments: point, prevPoint – points of the origin and the tip of the last robot's link, cylinders – array with description of environmental objects, n – number of objects in the array "cylinders". The function outputs the value "true" when there is a contact and "false" if no. Variable touchPoint is appropriated the value of the contact point, to variable "normal" - one of the normal. The block diagram is shown in Fig 4.25.

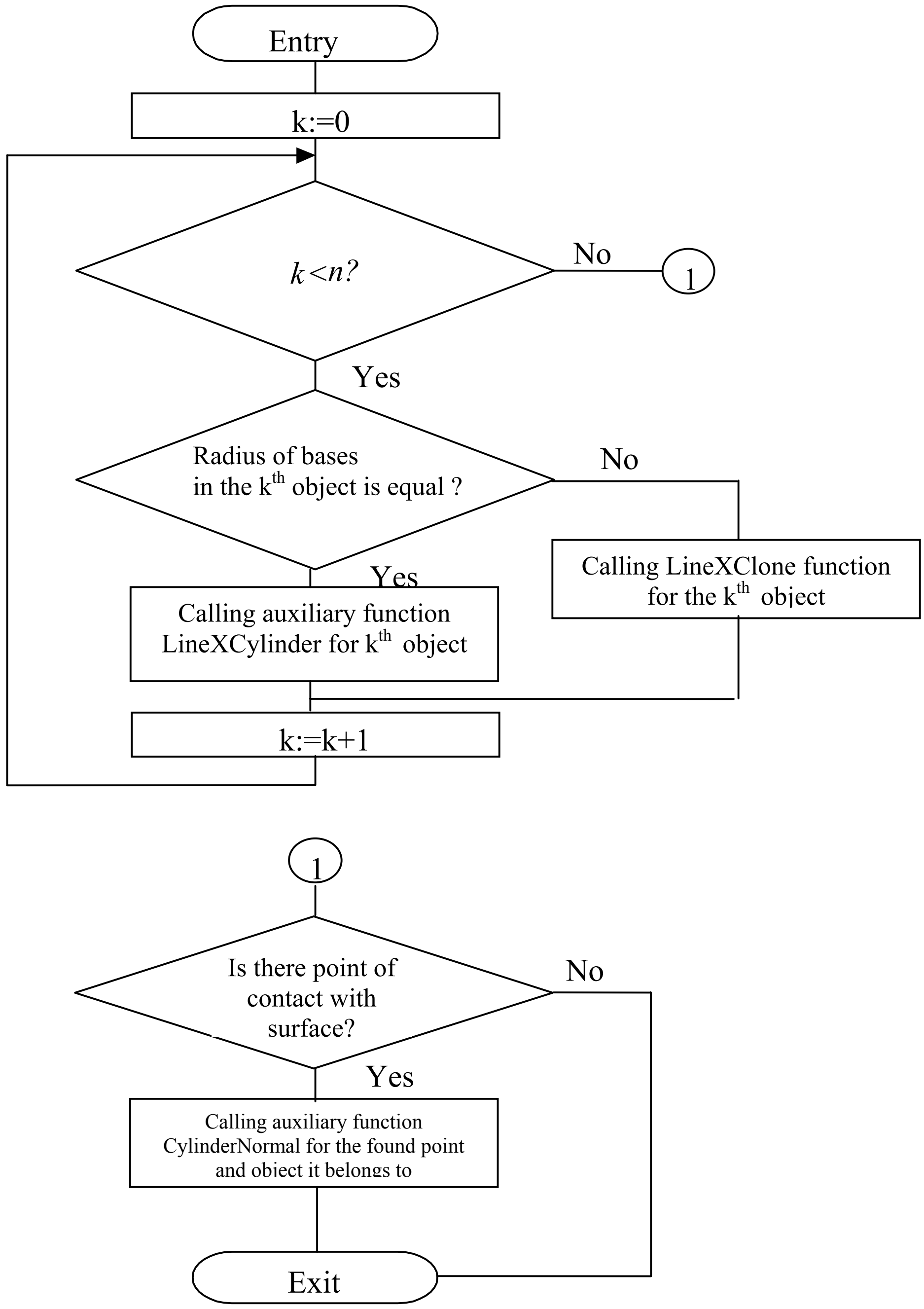


Fig. 4.25 Function FindCylindersNormal block diagram

Auxiliary functions

Function LineXCylinder(linePoint1,linePoint2:TPoint;cylinder:TCylinder; var xPoint:array of TPoint):integer; this function computes intersection points of line with a cylinder. Arguments: linePoint1, linePoint2 – coordinates of points the line goes through; cylinder – description of the cylinder. The function outputs the number of intersection points, computed coordinates of points are written in xPoint array. The block diagram is shown in Fig 4.26.

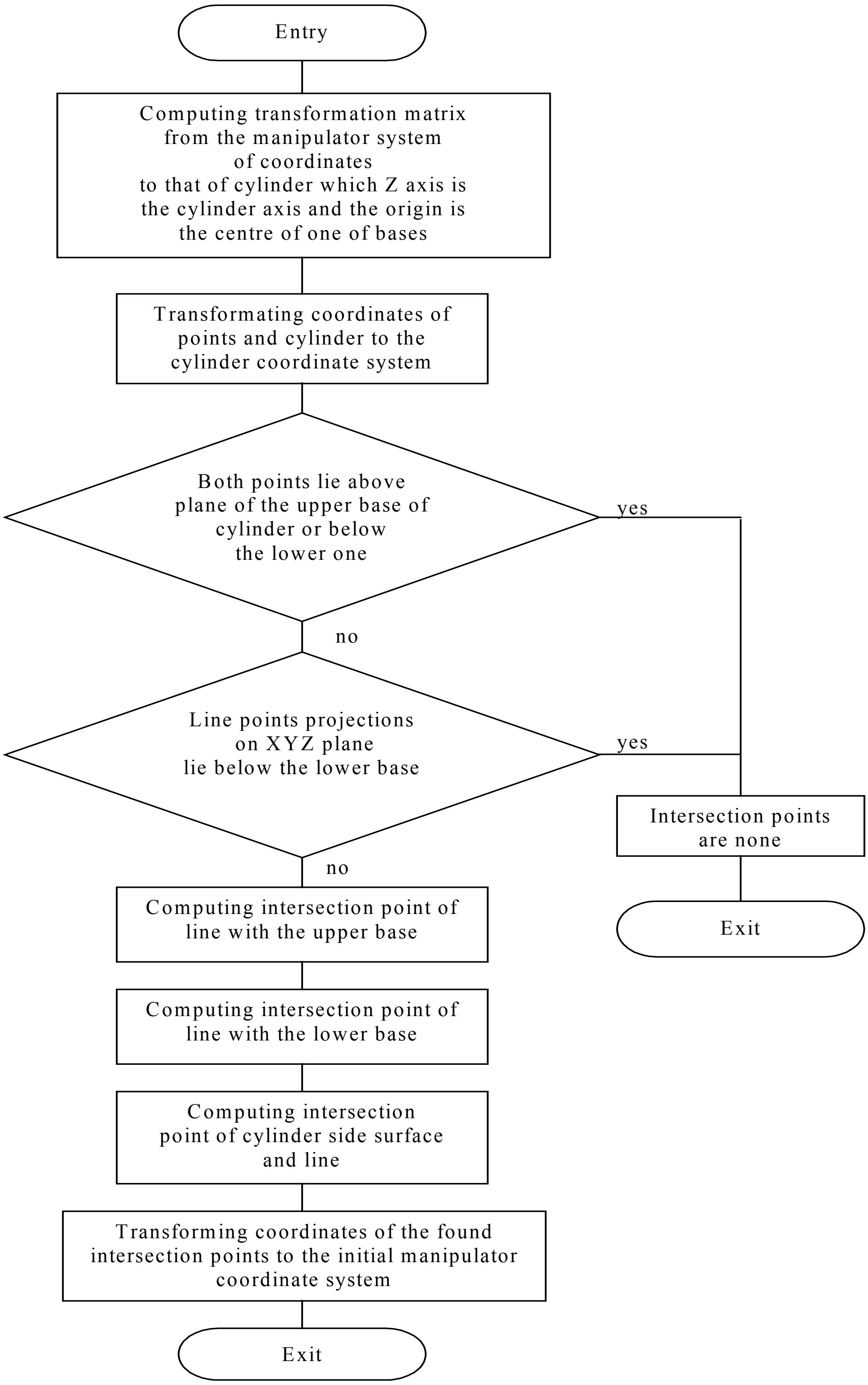


Fig. 4.26 Function LineXCylinder block diagram

Function LineXCone(linePoint1, linePoint2:TPoint; cone:TCylinder; var xPoint:array of TPoint):integer – the function computes intersection points with truncated cone. Arguments: linePoint1, linePoint2 – coordinates of points line goes through; cone – descriptor of cone.

The function outputs the number of intersection points, coordinates of points are written in xPoint. The block diagram is shown in Fig.4.27.

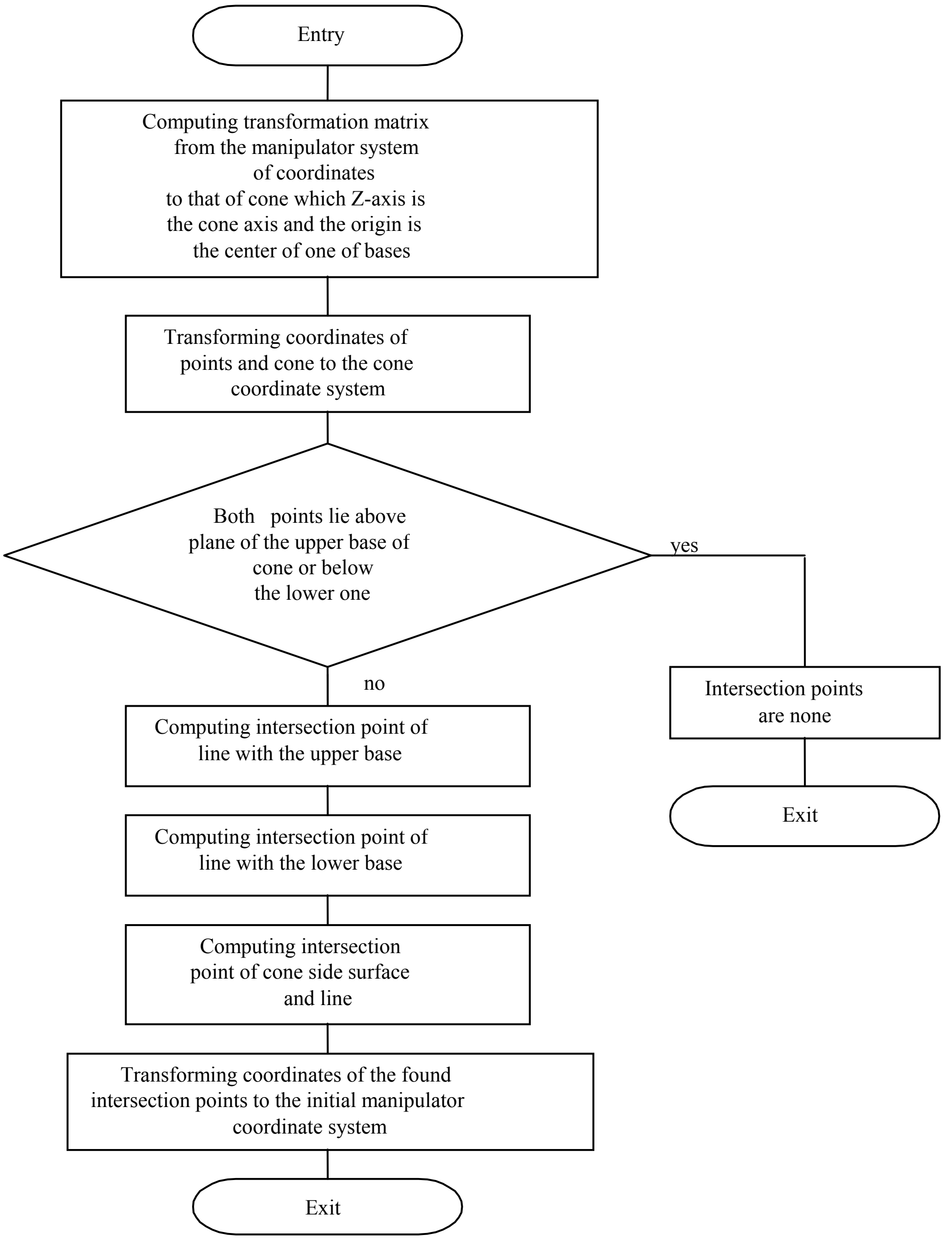


Fig. 4.27 Function LineXCone block diagram

Function CylinderNormal(point: TPoint; cylinder: TCylinder; var normal: TPoint): boolean – determines the normal to the surfaces of cylinder or cone in the point. The block diagram is shown in Fig. 4.28.

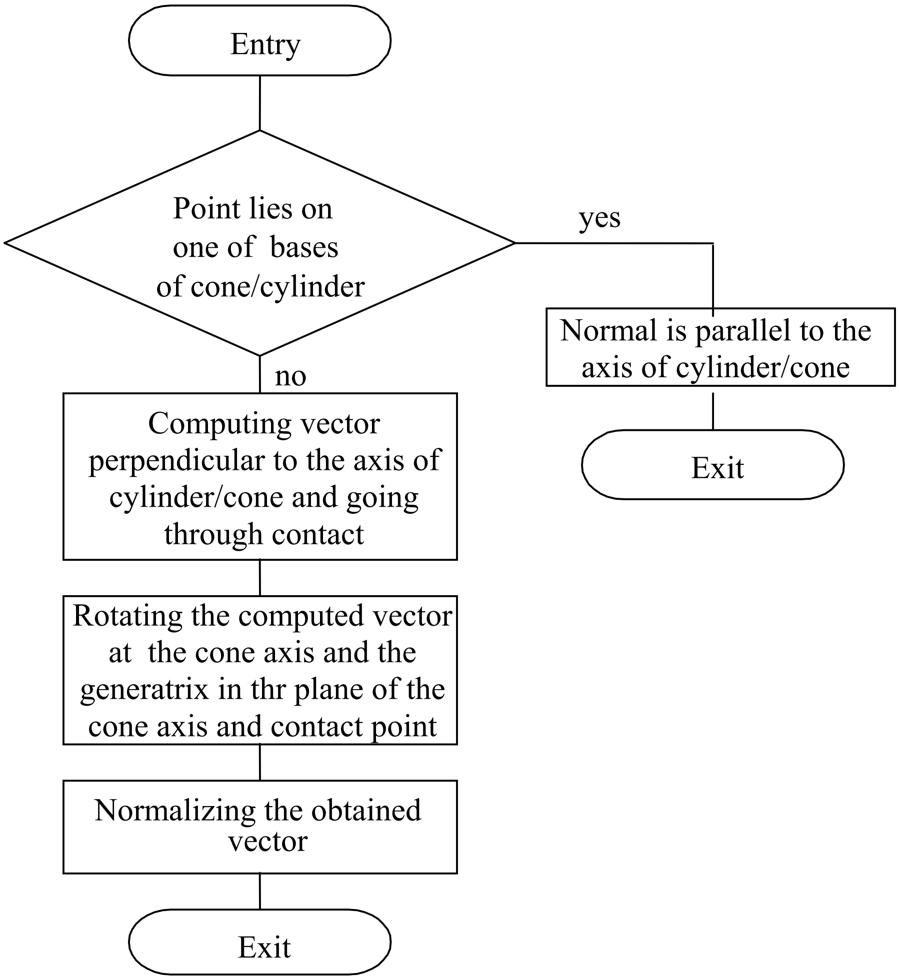


Fig. 4.28 Function CylinderNormal block diagram

Let's relate the software procedures and functions to block diagram Fig. 4.23 of the algorithm for detection of virtual manipulator's collisions with environmental objects.

Blocks 1, 2, 3 relate to procedure Execute.

Blocks 4, 12 relate to procedure CopyData.

Blocks 5, 6 relate to procedure me on Idle.

Block 7 relates to function CurrentManipulPaint.

Blocks 8, 9, 10, 11 relates to function Find↔CylinderNormal.

Chapter 5. Experimental study

The main goal of experimental testing of the developed technology for creating "augmented reality" is to establish a degree of realism to convince man, perceiving non-existent object acting on his visual and tactile and kinaesthetic senses, that it really exists in real environment.

A variant of "immersion" was experimentally tested in which man sees real world with the help of helmet displays showing picture taken with video cameras (videosee-through HMD).

The realism of visual perception, provided that one fulfills the effect of screening and identical illuminance of virtual and real objects etc., depends from two factors: continuity of perception and accuracy in registration of visible fragments of real environment with corresponding fragments of model ones.

<u>Continuity of perception</u> is determined with time needed for generating a frame of augmented reality image. This time must not be more than 50-60 ms what corresponds to the frame rate 15-20 Hz.

As it is, for generating a frame one needs:

- to take an image scene with the help of video camera, enter in the graphic station and digitize;
- to determine spatial coordinates of position and orientation of observer's head, which in their turn determine scale and aspect of generated virtual objects and model images of environmental objects, and enter them in the graphic station;
- to generate separately for left and right eye model images of environment and overlay them on real digitized images of the same augmented with virtual object's image and to show the combined picture on displays.

It is known, that a conventional video system operating at frame rate 50 Hz needs 40 ms for making a frame, the same time is needed for HTS' determining head's position-orientation coordinates.

Thus, the maximal attainable frame frequency cannot be more than 25 Hz but it is sufficient for perceiving picture as non-discrete one.

But three operations are to be completed for this:

- making a frame of real scene image;
- obtaining current coordinates of head's position and orientation;

- capturing video image to convert it in the digital form, generating digital image of virtual object to augment it to the real one (all this for left and right eyes) and showing the picture alternately to left and right eye.

For providing parallel execution of these operations the following devices are used:

- two mobile video cameras taking picture of real scene (Unit 1 in Fig.2.1) for left and right eye, realizing the first operation;
- computer of Unit 3 for computing coordinates of head's position/orientation, realizing the second operation;
- graphic stations 1 and 2 (Units 7 and 8) each of which incorporating two central processors operating in parallel, video capture card's processor and that of image video card.

To provide the required frame frequency each of the three operations must do its work in time ≤ 40 ms.

The surface of International Orbital Space Station (IOS) was used as geometrical model of real environment. Figs.5.1a, 5.1b present image of this model.

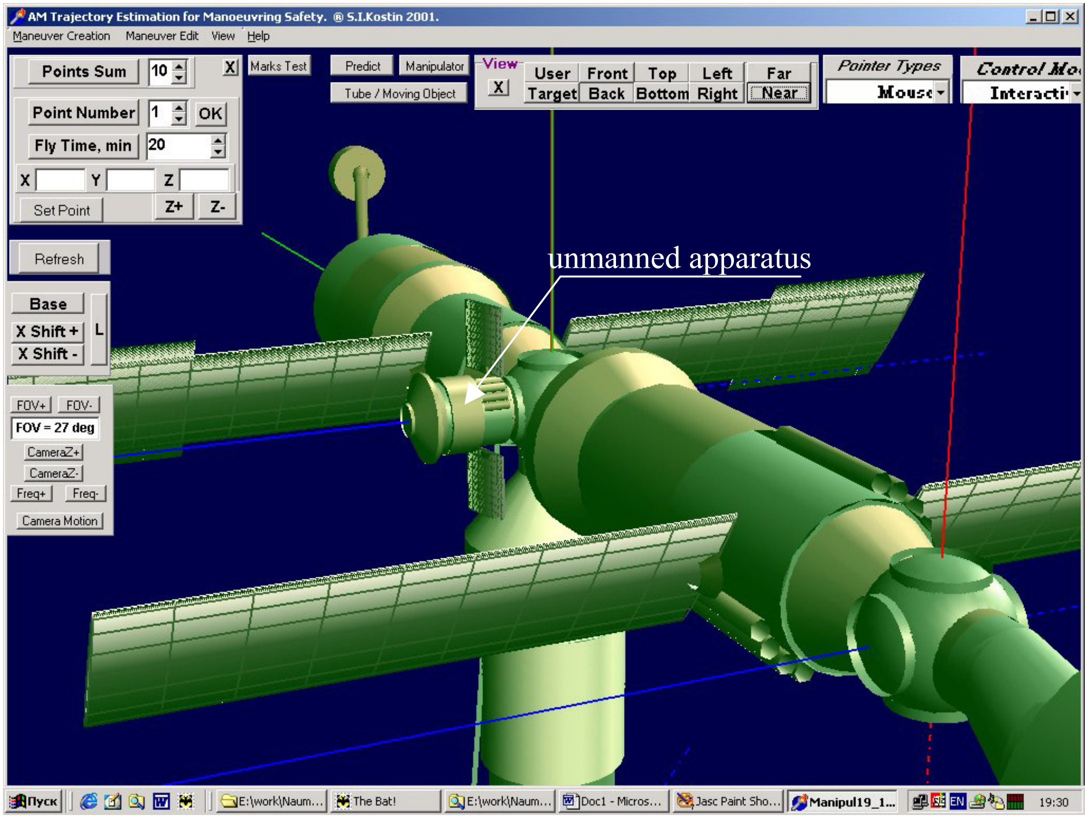


Fig. 5.1a 3D scene of IOS.

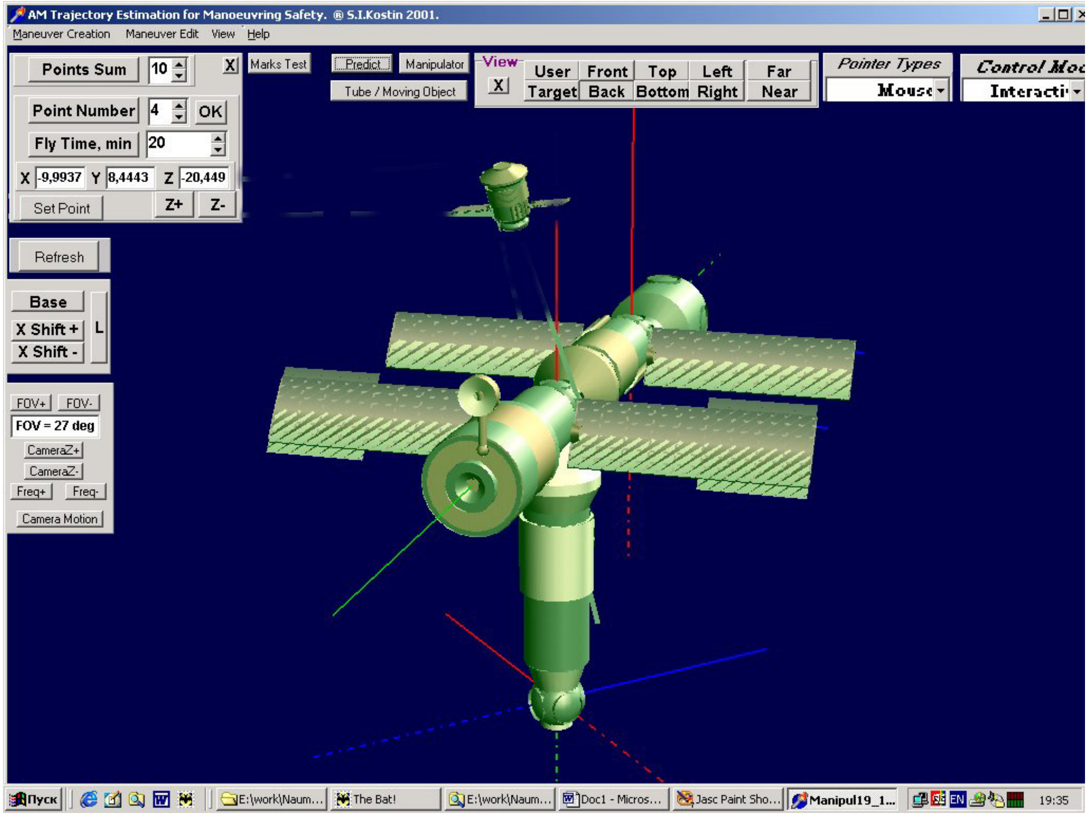


Fig. 5.1b 3D scene of IOS.

Fig.5.2 shows a wire representation of the geometrical model image.

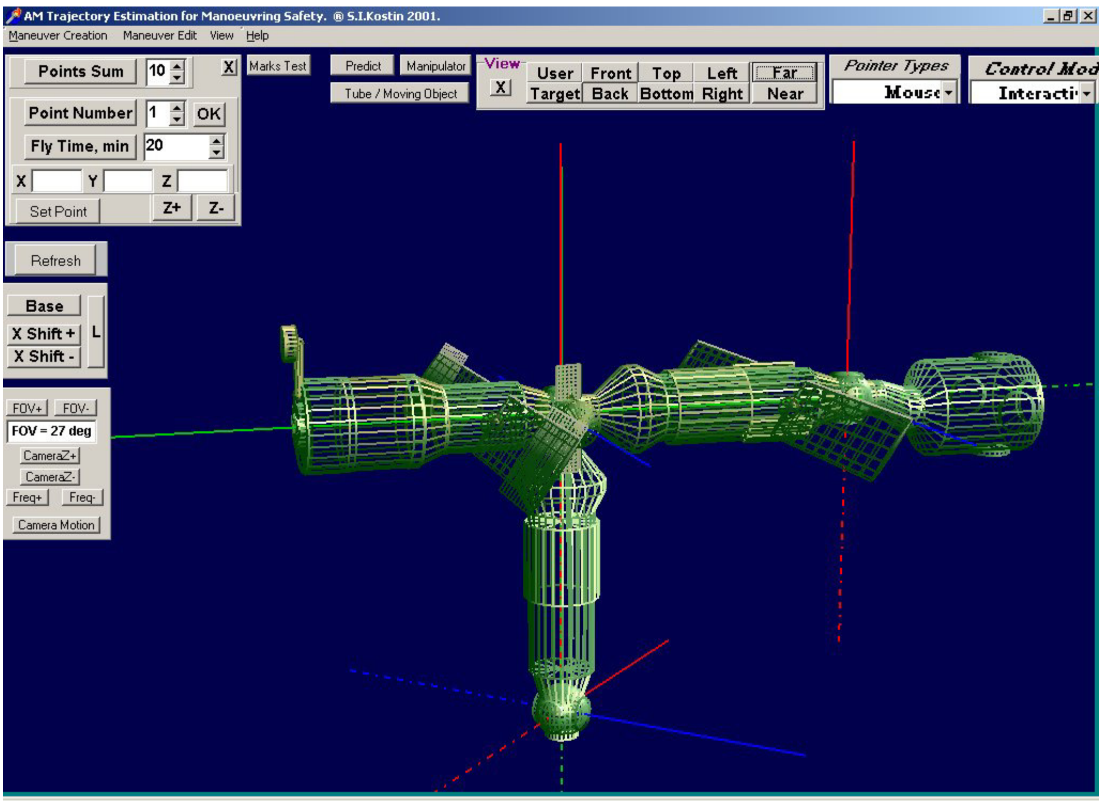


Fig. 5.2.   Wire representations of geometrical model images of IOC.

Figs 5.3a and 5.3b present the model of IOS assembly part (second variant of real environment) with the virtual manipulator.

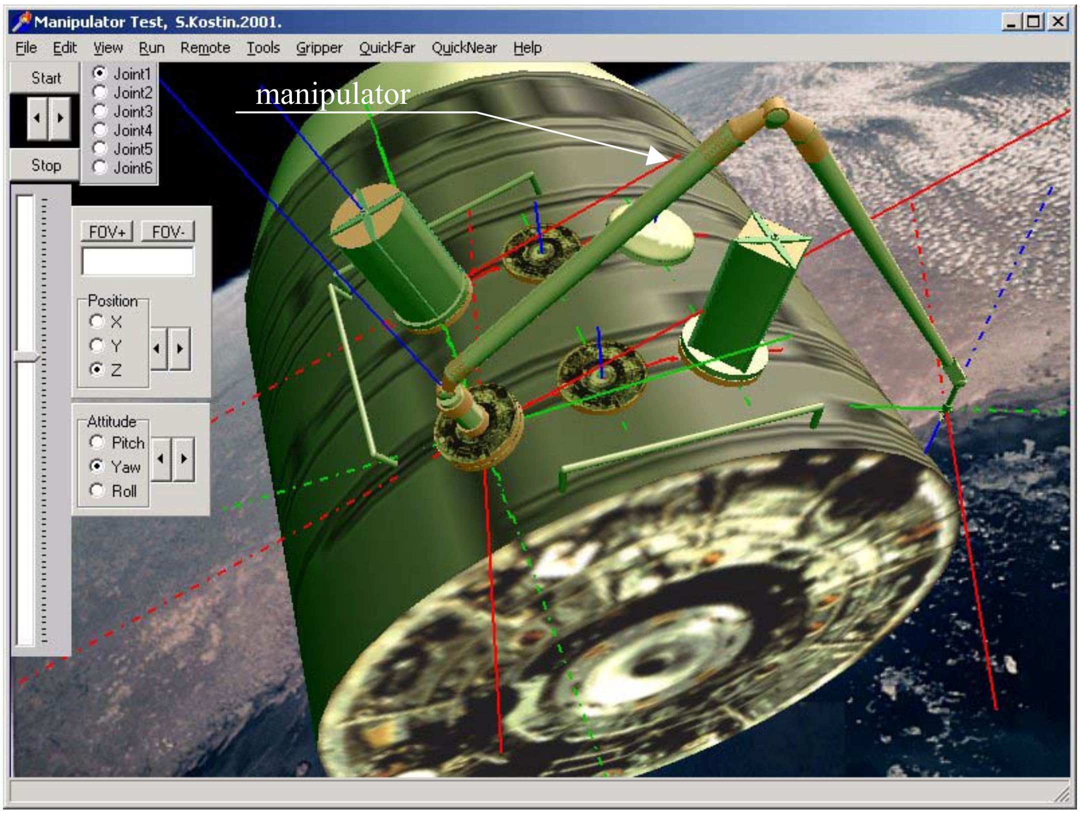


Fig.5.3a IOS assembly part and the robot-manipulator.

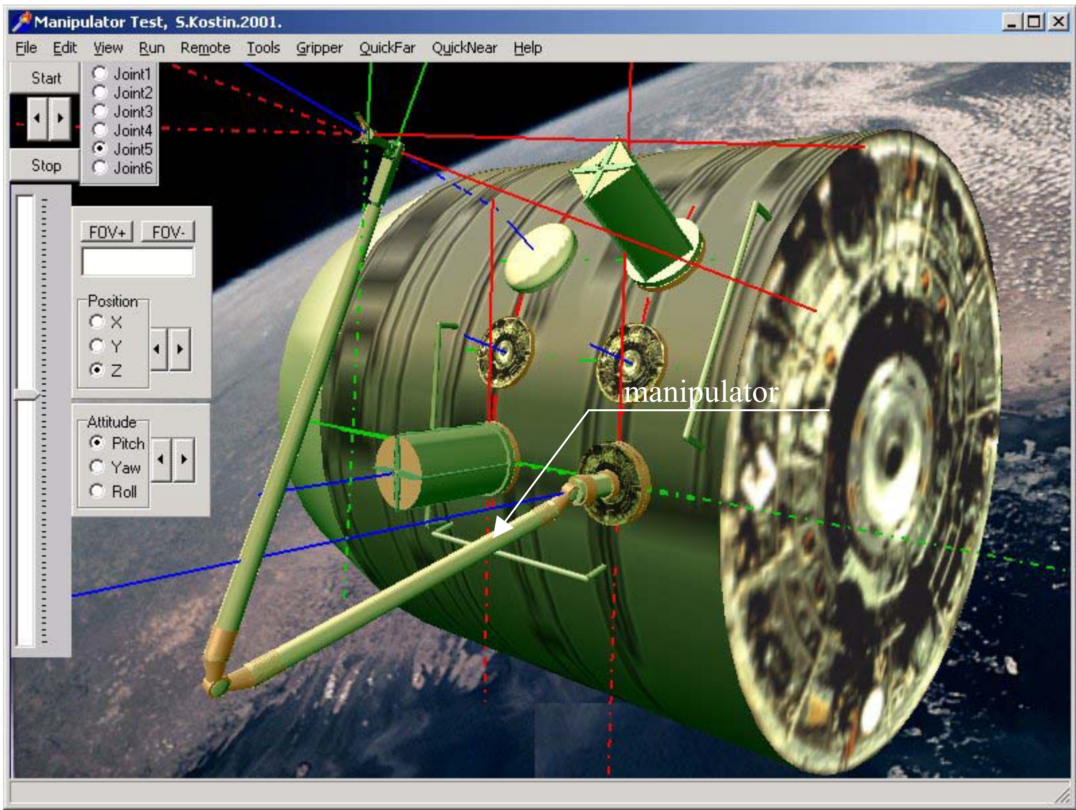


Fig. 5.3b IOS assembly part and the robot-manipulator

Fig.5.4 presents a wire representation this model image.

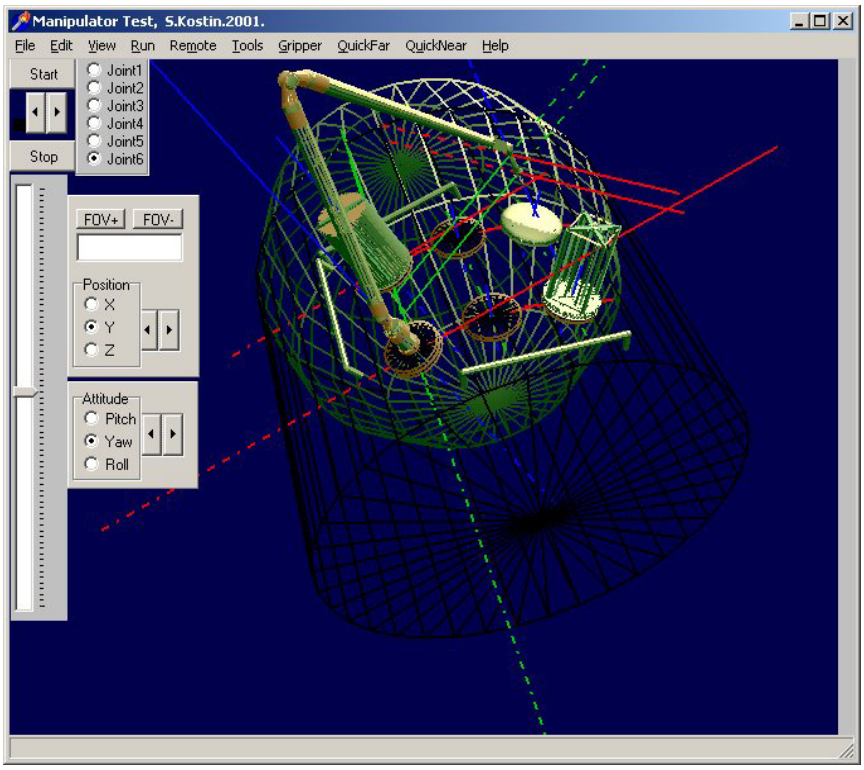


Fig. 5.4. Wire representations of geometrical model images of anthropomorphic robot

Image of this second real environment variant - Mock-up of the assembly part is shown in Fig.5.5.

Fig. 5.5 . Mock-up of IOS assembly part - real environment.

This mock-up was for making the augmented reality image.

Experiments for ascertaining the time for generating a frame and displaying its image have shown the following.

Using the graphic stations whose characteristics are given in section 4.1, for the case of real environment shown in Fig.5.5 and virtual manipulator shown in Figs. 5.3a, 5.3b we obtained the following times given in Table 5.1.

Table5.1

Expended time	Type of video card	
	AGP	AGPx2
Total time of frame generation, ms	39,5	35,1
Time of generating and displaying virtual objects image, ms	14,7	13,0
Time expended on environment's image capture, its digitizing, generating its geometrical model and displaying, ms	20,7	18,0
The rest	4,1	4,1

The rest time expenditures are lost in addressing the mouse, keyboard and receiving and processing data on head's coordinates etc.

If a more advanced card AGPx2 is used in place of AGP, time expenditures are less as one sees in the Table 5.1.

Table 5.2 presents the following detailed expended times in generating and displaying geometrical model shown in Fig.5.2.

Table5.2

Expended time	Wire model	Model with planes between wire ribs	Model with texture	Model with space background
Time of generating and displaying geometrical models images, ms	10	15	26	31

The performed experiments have proven a possibility of the required frame frequency. Two identical graphic stations that were used for that have the following characteristics: Pentium III 800 MHz, Chipset Intel's 440 Bx, Videoadapter ASUS-V6800, DDR AGP RAM32Mb, Videoinput AverMedia EZCapture, Monitor 19" Sony E400.

Precision of registering model and actual images of environment was also verified with the help of constructed Hard&Software complex facility.

As it is known, there are two kinds of registration errors - static and dynamic. Static errors are those that occur even if the observer's vision point and the position of the observed body remain entirely unchanged. Dynamic errors do not reveal themselves until the observer or the body of interest move. If one uses a helmet mounted display to observe a real environment with an Augmented Reality system, dynamic errors will be the principle component of the registration error.

It was cleared that main contributors to the ***static error*** are:

- nonlinear distortion of CCD camera;
- difference in FOVs (scales) of real and virtual cameras;
- error of measuring head coordinates leading to difference in position and orientation of the real camera, tracking operator's head (HTS), and the virtual camera taking model scene image;
- errors in constructing geometric model of environment.

The resulting action of the above causes leads to static errors up to 15 pixels using a camera with FOV 65°×65°. Changing this lens for one with FOV 40°×40° results in registration error 7-10 pixels owing to making less non-linear distortion of CCD camera.

For making less the difference in positions/orientations of real and virtual observation camera we changed the source of data input. The position/orientation data of the virtual camera, formerly taken from HTS computer, today are inputted immediately from the robot-like device which moves the real TV cameras. These data are position/orientation matrix of the real cameras on the end of the robot like device – its mobile platform. This matrix is calculated as a function of the robot-like device's joint coordinates, which are measured by position sensors in joints of the robot-like device.

To make clear the influence of errors, those in constructing the geometrical model of environment, on the resulting registration error it was tested using an environmental scene which geometrical model may be constructed with enough precision. A Cartesian system model was used as such a design, which is presented by Fig. 5.6.

However, even for this simple model the registration errors could not be excluded. Analyses showed that sources of the registration error are:

1) inaccuracy of the robot-like device's mathematical model which determines the position/orientation matrix T of the real cameras as a function of joint coordinates of the robot-like device;

2) the real camera's PSF center's non-coincidence with the lens' optical axis;

3) difference in FOVs of real and virtual cameras.

A special calibration technique is developed for exclusion of these causes.

This calibration serves to determine, so-called, correction matrix T_{corr} with whose help the matrix T^c is determined of actual position and orientation of the coordinate system fixed with real TV camera so that its origin being in the cameras' optical center and axis X_3 being the optical axis.

Matrix T is calculated as known function of joint coordinates. It is not equal to T^c as there is always errors in construction of geometrical model of environment. Therefore, it will be used for calculating T^c with the formula

$$T^c = TT_{corr}. \tag{5.1}$$

The proposed calibration method is based on determining position and orientation of real camera in the base coordinate system for a monitor image of some simple design whose position and orientation in the base coordinate system are precisely known. As such design may be one shown in Fig.5.6.

Then, using the known formulas of optical transformation, we find:

$$x_1^{(j)} = -k_x f \frac{T_{1.}^c x^{(i)}}{T_{3.}^c X^{(i)}} \quad i=1,2,...,n,$$

$$x_2^{(i)} = -k_y f \frac{T_{2.}^c x^{(i)}}{T_{3.}^c X^{(i)}} \quad i=1,2,...,n, \tag{5.2}$$

where $x_1^{(i)}$, $x_2^{(i)}$ - are coordinates of the i-th characteristic point of images on the monitor's screen in the cameras' coordinate system,

$X^{(i)}=(X_1^{(i)},X_2^{(i)},X_3^{(i)})$ – is the positional vector for the i-th characteristic point of the design in the base coordinate system,

$T_{1.}^c, T_{2.}^c, T_{3.}^c$ - are the first, second and third lines of the unknown matrix T^c of position and orientation of the camera's coordinate system in the base coordinate system (system of observation) which is needed for generating models of virtual object and environment.

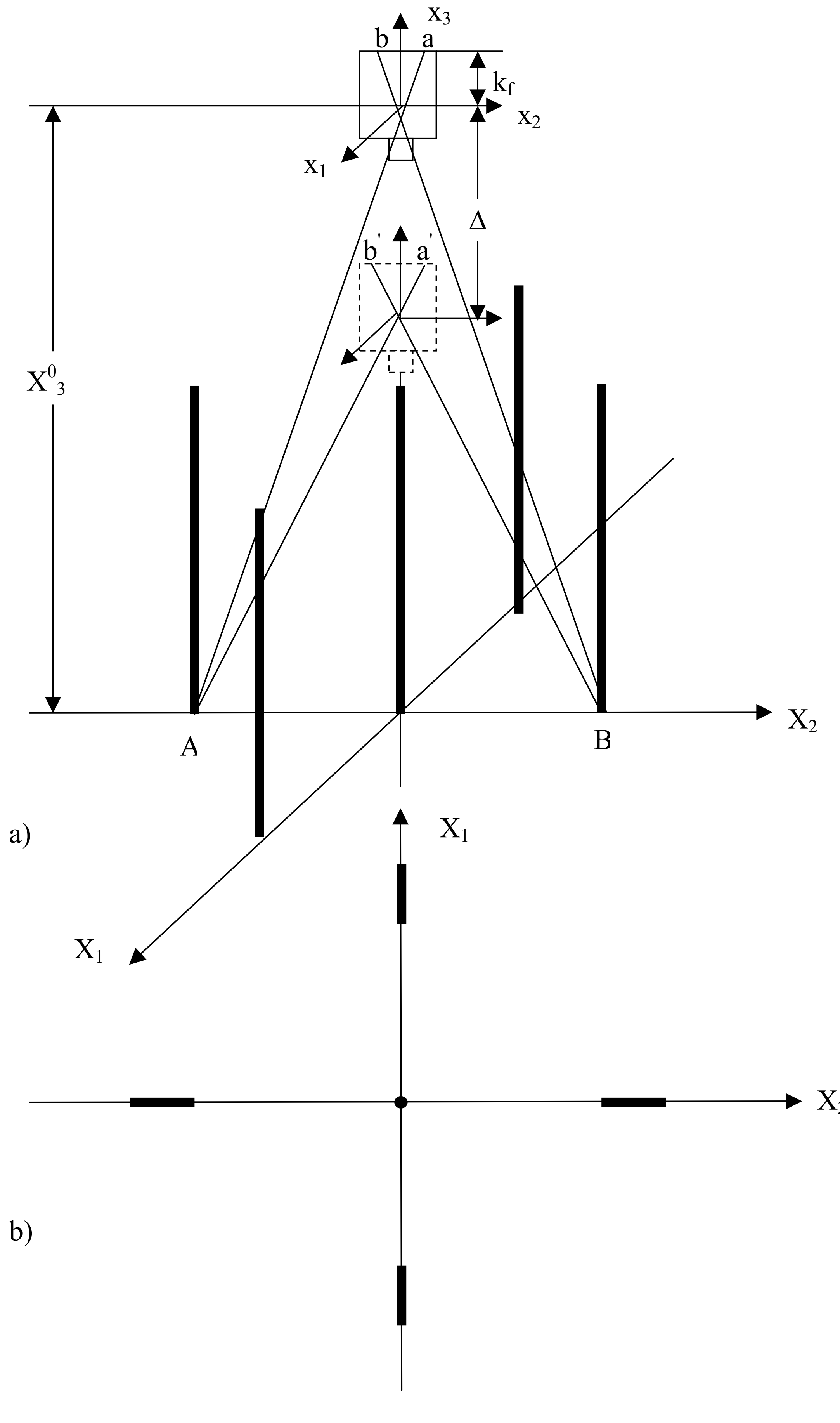


Fig.5.6 Special design (Cartezian system model) for definition of calibrationmatrix T_{corr} (a) and TV image of the special construction for position/orientation TV camera determined by matrix T_0^{r} (b).

Evidently, 12 equations (5.2) are enough, generally, for determining all elements of the unknown matrix and two more for finding unknown quantities $k_x f$ and $k_y f$.

Thus, generally, 7 characteristic points are enough for accomplishing the task of finding the unknown matrix T^c by solving the system (5.2).

In practice, the task may be made simpler if one so gives position and orientation of some known design in the base coordinate system that characteristic points would have one or two zero components and, also, be symmetrical relative to the axes. That will make system (5.2) simpler.

The found current matrix T is to be multiplied at the right side by T_{cor} for obtaining the current matrix of virtual camera's position/orientation.

The matrix T_{cor} may be determined simpler if matrix $T_0{}^r$ entering its expression (5.2) is assigned rather than computed. And it is expedient to choose such value for it which corresponds to a position/orientation of TV camera easily attainable with hand control of the robot-like device, e.g. by way of successive translations along the axes of the base coordinate system and then rotations round them.

Such a matrix may be that of a form:

$$T_0{}^r = \left\| \begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & X_3^0 \\ \hline 0 & 0 & 0 & 1 \end{array} \right\| .$$

If one takes the design shown in Fig.5.6 as the calibration construction imaged with the TV camera and assumes axes X_1, X_2, X_3 of this design to be the base system's axes, then the TV camera's position will be defined with the matrix $T_0{}^r$ and the design's image will have form shown in Fig. This images will be similar at any camera's distance X_3 from the construction and differ only in scale. For determining the unknown X_3 the following method is expedient. Let us measure distance d_{AB} between images a and b of some characteristic points in the design, e.g. A and B lying on a known distance D_{AB} along the axis X_2 of the base system when the TV camera occupies the position $(0,0,X_3{}^0)$. Then we will move the camera along X_3 at distance Δ and again will measure the distance d'_{AB} between images a' and b' of points A and B. Then equations of optical transformation will give

$$d_{AB} = \mathrm{Kf}\frac{D_{AB}}{X_3^0}\ , \qquad d'_{AB} = \mathrm{Kf}\frac{D_{AB}}{X_3^0 + \Delta}\ .$$

The unknown X_3^0 will be determined as $X_3^0 = \dfrac{\Delta \cdot d'_{AB}}{d_{AB} - d'_{AB}}$.

Having determined this way T_0^r and knowing matrix T found for known values of robot-like device's joint coordinates, one can with the help of () determine value of T_{cor} as

$$T_{cor} = T^{-1}\, T_0^r .$$

The computed matrix T is to be multiplied by this matrix at the right side to obtain current matrix of virtual camera's position/orientation.

The error caused by misalignment of FOVs of real and virtual cameras can be eliminated by a manual adjustment of the virtual camera's FOV.

This adjustment is provided for in SW for generating augmented images and activated with the keyboard or mouse. An image of the abovementioned calibration construction is displayed on the monitor for that and overlaid with that of virtual one having the same scale and generated for the same position/orientation of virtual camera as real had. An error in registration of the images is caused by misalignment of FOVs of real and virtual cameras.

Below, a row of figures are presented which illustrate the different variants of augmentation of virtual object (manipulator) image to that of real environment obtained with the TV camera.

Figs. 5.7 and 5.8 illustrate the real environment TV image which is mock-up of assembly part of IOS and this real environment augmented by virtual manipulator.

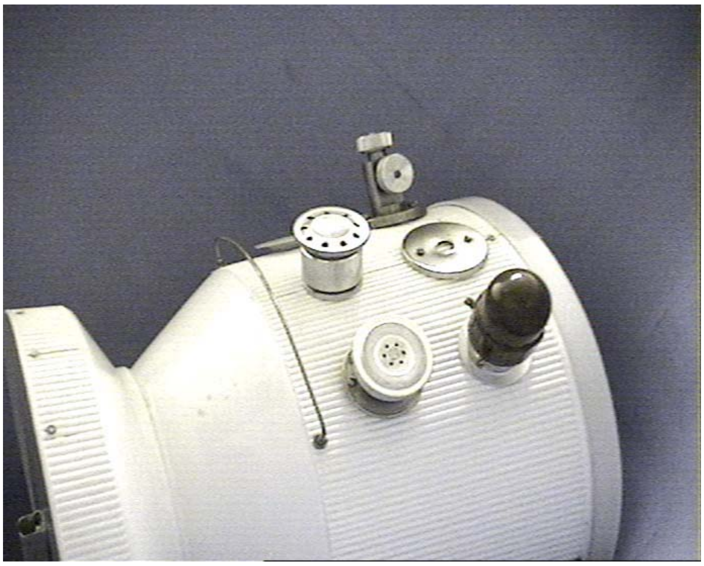
Figs. 5.9 and 5.10 illustrate the same but another aspect.

Figs. 5.11-5.13 illustrate the process of augmentation.

Figs. 5.14-5.17 illustrate the augmented reality images if the virtual manipulator is moved.

Figs. 5.18-5.21 illustrate the augmented reality images if man-observer is got nearer to mock-up of assembly part of IOS.

Figs. 5.22 illustrates a complex effect of virtual manipulator's screening the scene and otherwise.



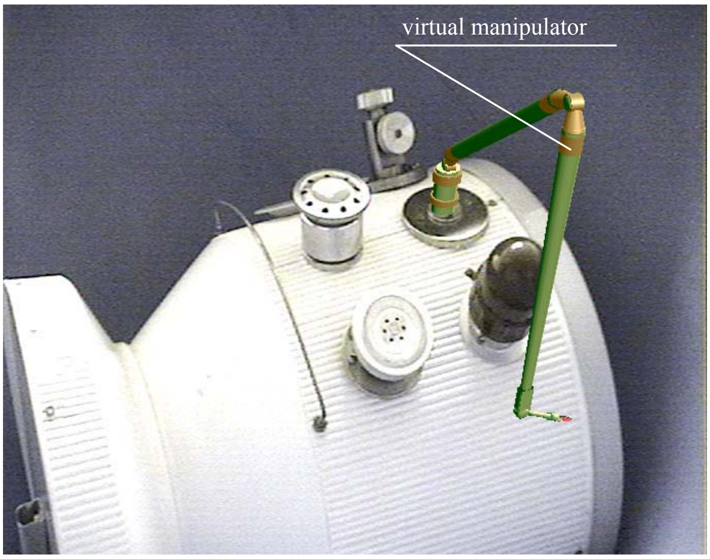
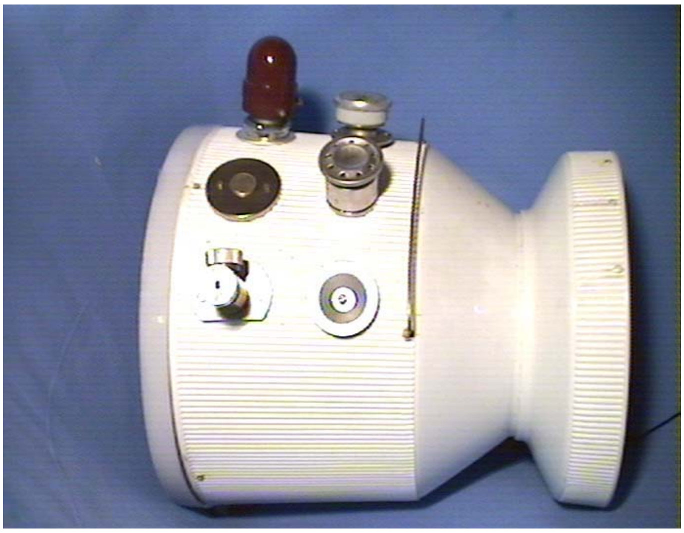


Fig. 5.7 . Real environment TV image mock-up of assembly part of IOS.

Fig. 5.8. Real environment augmented by virtual manipulator



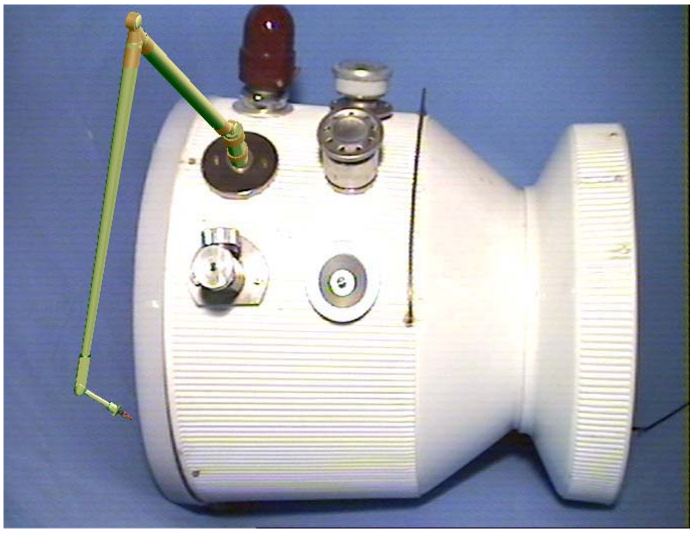


Fig. 5.9. Real environment TV image mock-up of assembly part of IOS.

Fig.5.10. Real environment augmented by virtual manipulator

The errors' being different over the FOV showed that the main remaining cause of registration error is non-linear distortion. Figures 5.11-5.13 illustrate the process of augmentation of virtual object's (manipulator's) image to that of real environment obtained by the camera.

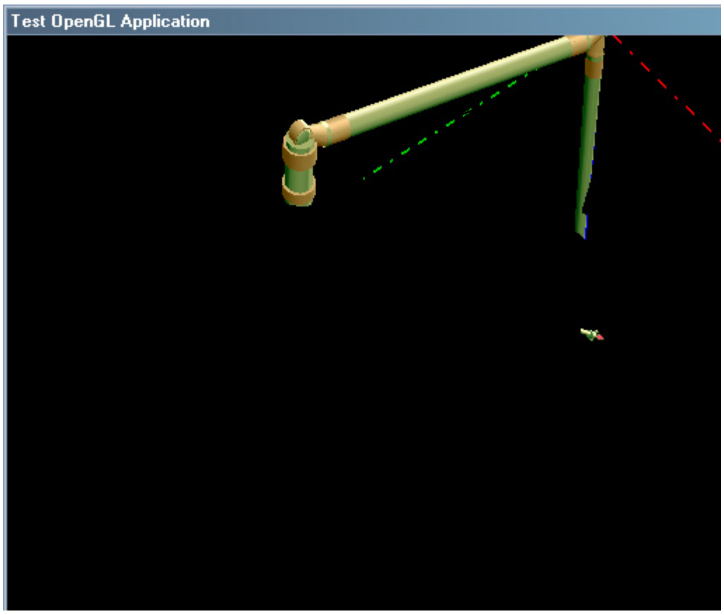


Fig.5.11. Image of unscreened parts of virtual object (manipulator) – mask.

Fig.5.12 Image of real environment with the mask.

Fig.5.13. Image of real environment augmented with the virtual manipulator.

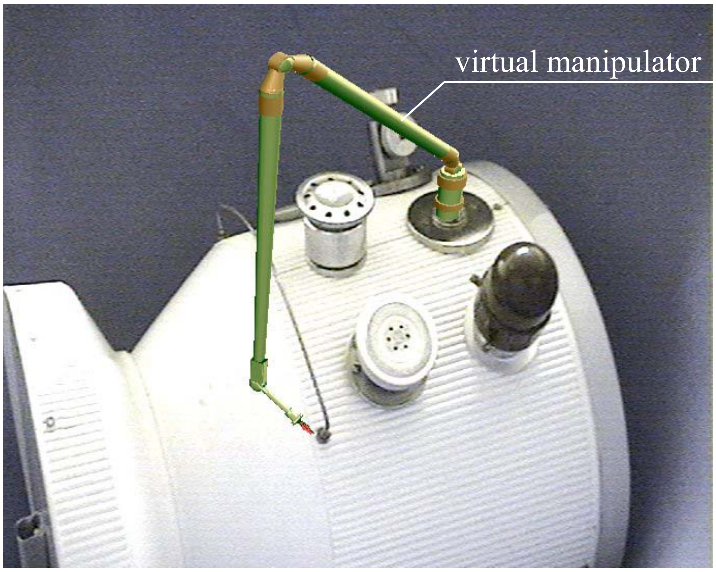


Fig.5.14 Illustration of the moving virtual object (I phase).

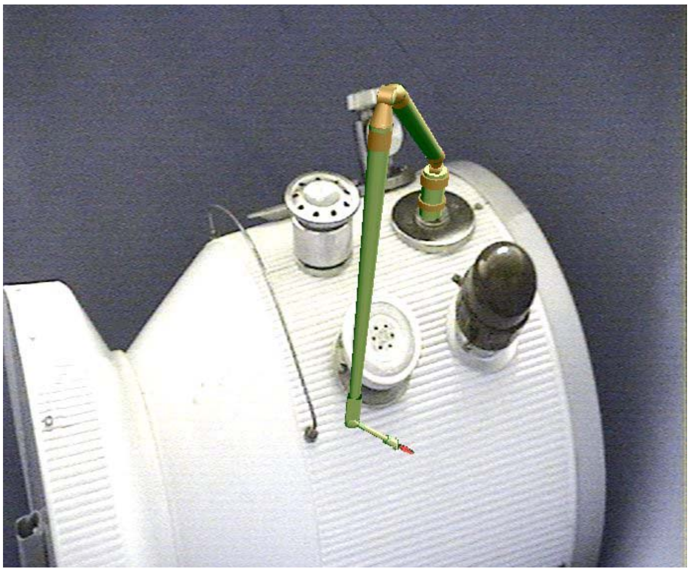


Fig.5.15 Illustration of the moving virtual object (II phase).

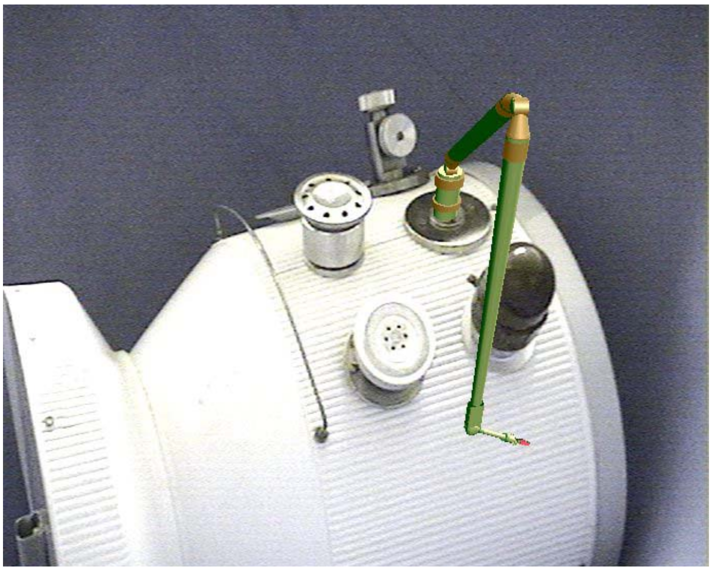


Fig.5.16 Illustration of the moving virtual object (III phase).

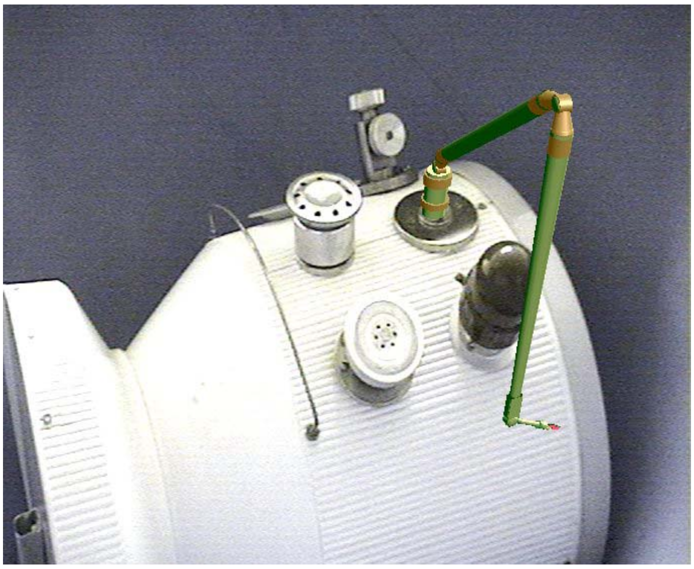


Fig.5.17 Illustration of the moving virtual object (IV phase).

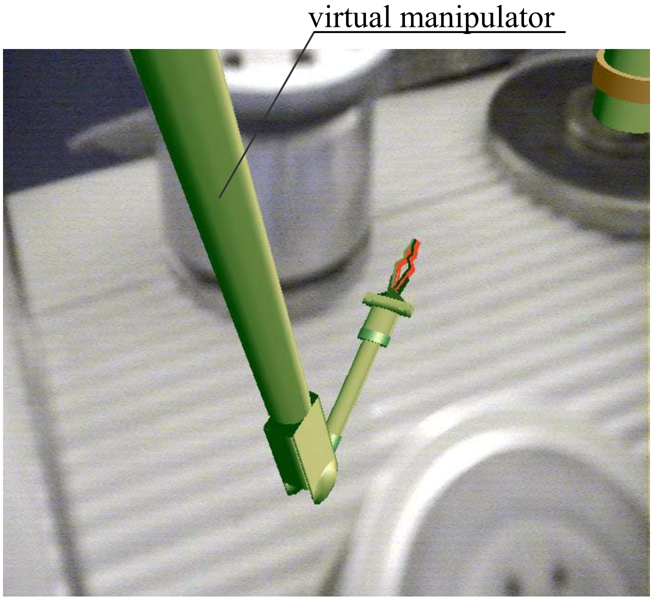


Fig.5.18 Illustration of the Augmented Reality images if the man-observer moves (I phase).

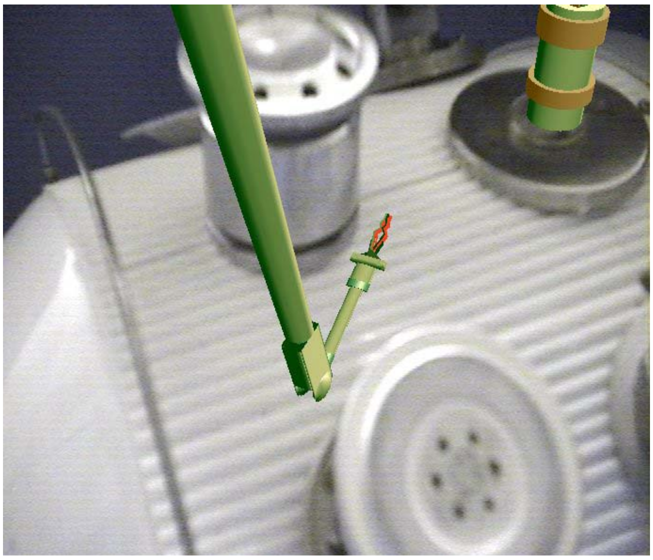


Fig.5.19 Illustration of the Augmented Reality images if the man-observer moves (II phase).

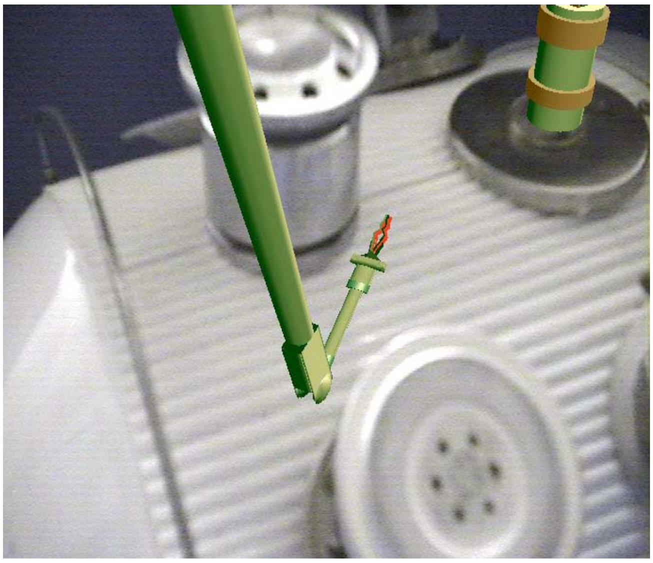


Fig.5.20 Illustration of the Augmented Reality images if the man-observer moves (III phase).

Fig.5.21 Illustration of the Augmented Reality images if the man-observer moves (IV phase).

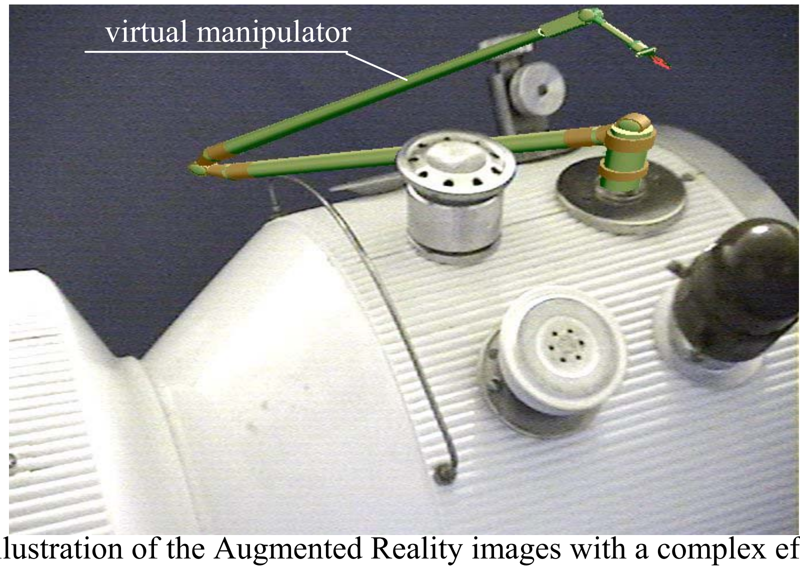
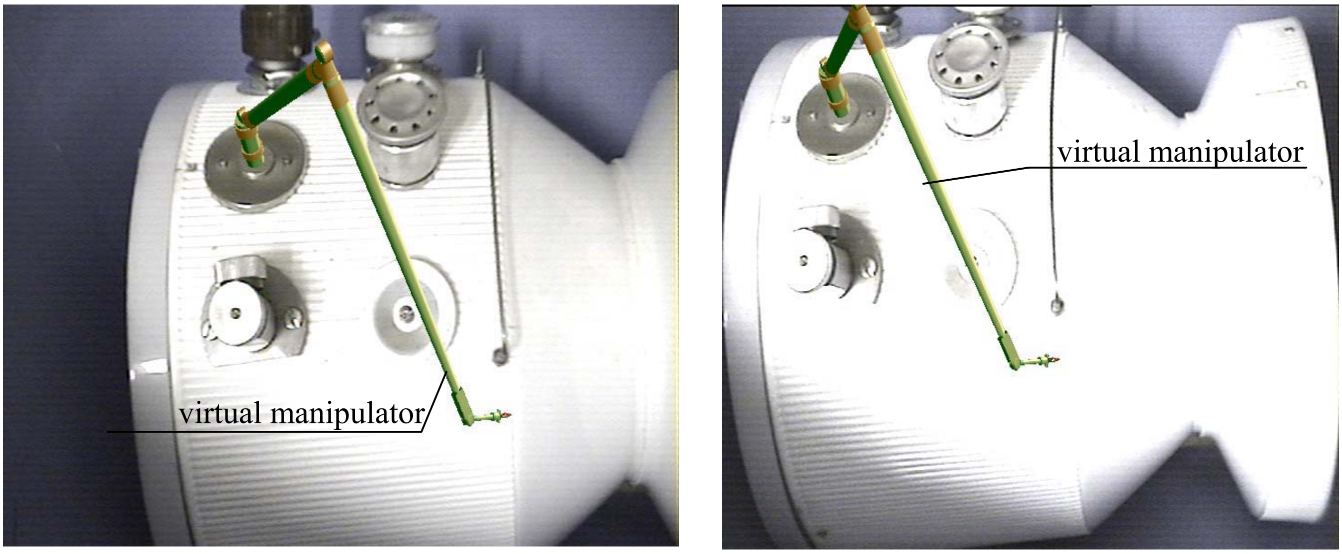


Fig.5.22 Illustration of the Augmented Reality images with a complex effect of screening.



a) b)

Fig.5.23 Illustration of the Augmented Reality images for left (a) and righ (b) eyes.

The dynamic error, when observer moves his head, appears as difference in images of real scene imaged with camera and that of geometrical model obtained with the help of computer synthesis. This difference is caused:

- by difference in positions and orientations of real camera, at which scene is imaged, and of virtual camera, at which the model image is generated;
- by image's shift (blurring) for time of exposition.

As experiments showed, the major cause of the difference in positions and orientations of real and virtual cameras arising from operator's moving his head is the dynamic error of robot-like device's tracking head's position and orientation. It is caused with slowness of the robot-like device and its imperfect control.

The utilized control system "Sfera-36" is very fast. It tracks step input exponentially with time constant 0,07 s. But such dynamic characteristics appear to be insufficient. Actually, with such a time constant at head's turning with angular speed 40 deg/s the settled tracking error will be 2,8 °. At the camera's FOV 40 ° such an error corresponds to a significant virtual image's shift relative to that of real. This shift will be 7 % of the screen what is absolutely unacceptable if one takes in account that control signal comes from the data processing computer of Unit 3 with the time delay 40 ms which is time expended on computation.

A possible way to reduction of difference between real and virtual cameras' positions and orientation is a more dynamic control system but the experiment showed that it can give only halving of the error what is not sufficient.

The more radical way is to use for the scene model's synthesis not coordinate data obtained with head tracking system (HTS) but those of current position of the observation camera. They are computable for the measured joint coordinates of the robot-like device as the matrix of position/orientation of the platform bearing the cameras as a known function of these coordinates.

Just this way was chosen for developing a prototype of the Soft&Hardware Complex Facility for testing the proposed technology for virtual body's immersion in real environment.

Yet this approach does not take away the aforementioned time delay of tracking by platform of head position/orientation caused with time expenditure of head space coordinates' calculation executed by HTS computer (Unit 3) and inertia on robot-like-device. But, as experiments showed, man practically does not perceive this delay.

The blurring of video image is the more the more is the camera exposition and this spoils realism of virtual body's "immersion" in real environment[1].

First, it is owing to the principal impossibility to register a blurred video image with indistinct contour and the clear computer-synthesized one. Second, it appears difficult, in this case, to determine optimal data of virtual camera's position/orientation to which the synthesized image is to correspond because the exposition varies from frame to frame depending from illumination, still not coming out of 40 ms of the frame generation time.

The optimal moment is the middle of exposition time. But its determination is complicated with the time spent on measuring the robot-like device's joint coordinates, transforming them to those of the camera, transmitting them to graphic station for generating the computer image. But even knowing it will not completely solve the problem of precise registration of virtual and computer images for moving observer.

An effective way to lessening the blurring is to utilize special cameras with short exposition what, in principle, opens a possibility of its radical reduction and more precise determination of virtual camera's position. But practical realization of this approach needs a considerable technical modification of the hard&software complex realizing the technology of immersion.

The main modification is one providing the synchronism of the following moments: beginning of the scene's image frames (observation cameras), beginning of the reference device's image frames (HTS), beginning of the scene model's image frames (generated with the graphic stations, Unit 7 and 8) and, finally, the moment taken for determining position of the virtual camera (central processor, Unit 2). Moreover, the cyclic operations of algorithms of Unit 2 and Units 7 and 8 must comply with the period of video image frames.

[1] Note, that in the exposition time 20 ms (50 % of the frame generation time) head turning with speed 40 °/s makes 0,8 °, i.e. about 2 % of the monitor's screen.

Such modification will provide a rigid temporal succession of operating algorithms of different units, identical on each period of video frame. This, in its turn, will make easier the extrapolation (prediction) of head's and observation camera's position/orientation which is indispensable because the time interval for generation of the position-orientation matrix is comparable with the time for making a video frame and during this time the observation camera may go far.

Fig.5.24 shows a temporal diagrams of each unit's operation explaining a possibility to reduce the dynamic error of registration of real environment image and computer-synthesized image of its geometrical model.

The mobile cameras for scene observation (Unit 1) and cameras imaging the reference marks (HTS) in time t_k begin to generate their images. And the position/orientation of the cameras precisely corresponds to the moment t_k because time of exposition is comparatively insignificantly small.

The computer of Unit 3 generates t_k the value $T^{H}(t_k)$ of the head's position matrix for the moment and computes the desired position matrix $T_d^{C}(t_{k+1})$ of the observation cameras for the moment (t_{k+1}) extrapolating the $T^{H}(t_k)$ to the moment (t_{k+1}).

This value is passed to Unit 2 (mobile cameras' control system) as the target value to be tracked by the platform bearing the cameras.

Because the value t_{k+1}- t_k=Δt is known at the proposed approach, the executed value of the matrix T^{C} in the moment t_{k+1} is determined with the formula:

$T_d^{C}(t_{k+1}) = T^{H}(t_k)\delta T_d^{H}(\Delta t)$,

where $\delta T_d^{H}(\Delta t) = \begin{bmatrix} 1 & -\delta_3 & \delta_2 & \Delta_1 \\ \delta_3 & 1 & -\delta_1 & \Delta_2 \\ -\delta_2 & \delta_1 & 1 & \Delta_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ - is incremental matrix for interval t_k-t_{k-1}=Δt,

Δ_1, Δ_2, Δ_3, δ_1, δ_2, δ_3 are known small translation and rotation of the head's coordinate system relative to the base one for time t_k-t_{k-1}=Δt equal to interval t_{k+1}- t_k=Δt.

This formula is true, if to consider that for equal adjacent small time intervals increments of the matrix on them will be equal. It is implied, also, that the computer's efficiency will enable all necessary computations in time interval Δt.

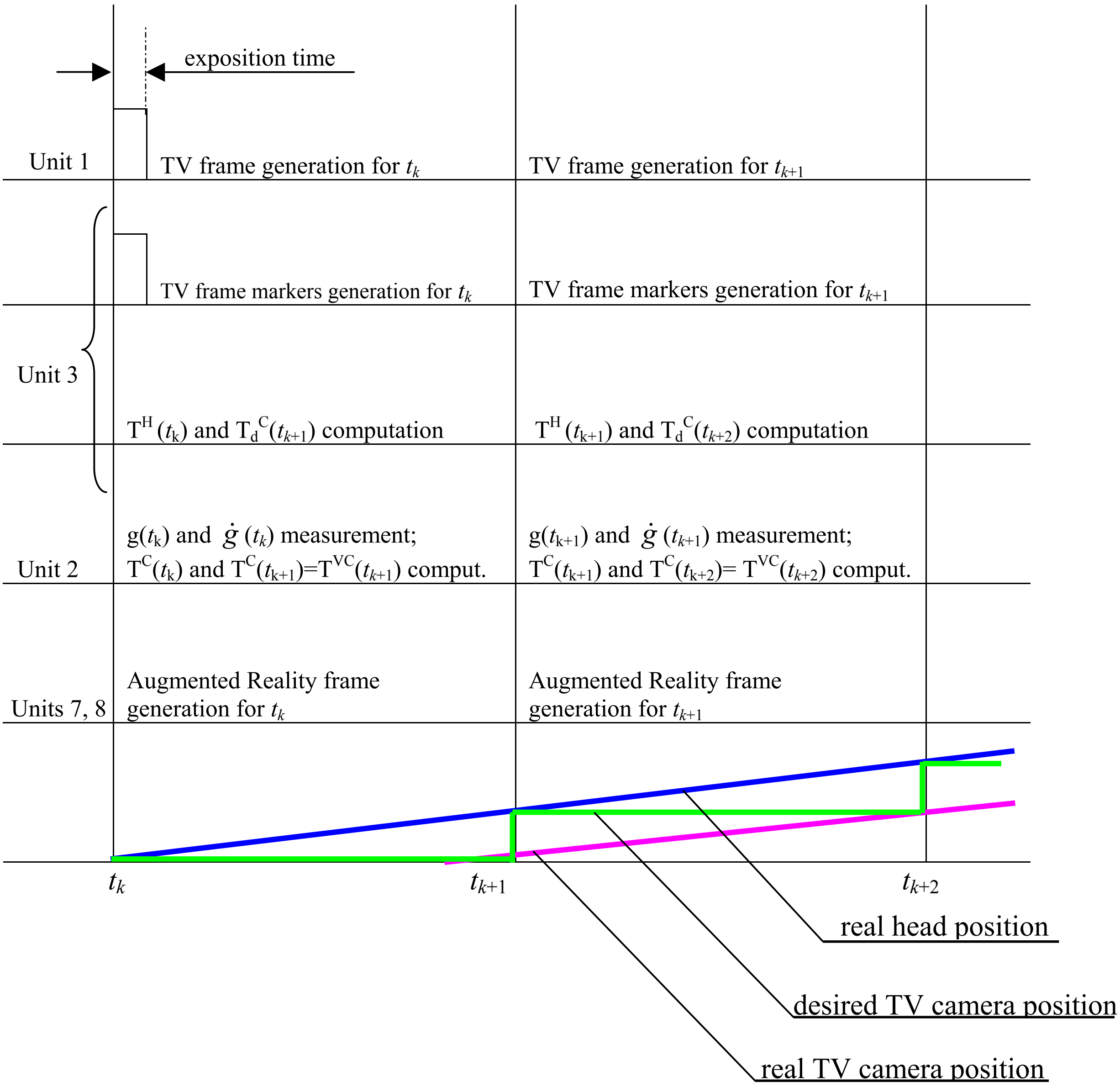


Fig.5.24 Operational time diagrams of experimental facility units.

Unit 2 provides sampling in the moment t_k of joint coordinate sensors $g(t_k)$ and joint speed sensors $\dot{g}(t_k)$ (it is supposed that sampling time is small) and computes for these values, solving the direct geometric and kinematic tasks, current observation camera's position matrix $T^c(t_k)$ and matrix $\dot{T}^c(t_k)$, whose non-diagonal elements are speeds of changes in elements of matrix $T^c(t_k)$ in the moment t_k and the diagonal ones are units, and, also calculates , matrix $T^c(t_{k+1})$ of matrix' T^c expected value in the moment (t_{k+1}).

The matrix $T^c(t_{k+1})$ must be equal to that of virtual camera's position/orientation $T^{vc}(t_{k+1})$ to which the frame must correspond, which is generated for the moment (t_{k+1}) with Units 7 and 8, of real scene image augmented with virtual manipulator.

The matrix $T^c(t_{k+1})$ is computed with the formula:

$$T^c(t_{k+1}) = T^{vc}(t_{k+1}) = T^c(t_k)\ \dot{T}^c(t_k)\,\Delta t,$$

which implies a realistic hypothesis, that speed $\dot{g}(t_k)$ in the interval Δt is constant.

Thus, in the moment t_{k+1}, repeated every video frame, the synthesis begins of the computer image frame representing the geometric model of real scene. Its scale and aspect is identical to those of the real camera with uncertainty complying with a degree of probability of the accepted assumption: camera's speed on the interval $[t_k, t_{k+1}]$ is constant. This makes possible to reduce the dynamic error of registration to the minimal value.

An obvious disadvantage of the proposed approach is a requirement of high illumination of work scene without which the exposition cannot be small. A possible alternative is utilization of pulsed sources synchronized with video frames.

Unfortunately, we couldn't realize this approach on the constructed Hard&Software Complex Facility.

The cycle of the Unit 2 algorithm equal to 32 ms does not comply with those for imaging work scene and HTS' reference marks. This fact made coordination of cycles for algorithms of Unit 2 and Unit 3 impossible what made utilization of cameras with short exposition unreasonable.

In the existing publicated version of the Hard&Software Complex Facility for testing the proposed technology of "immersion" the head rotation speed is not to exceed 2-3 deg/s for acceptable dynamic registration error of model and video images of scene. And with that, the dynamic registration error ≤ 6-8 pixels.

The tactile-kinaesthetic aspect of immersion was experimentally studied both for a case of virtual solid body and telecontrolled manipulator.

The following devices were used for the first case (Fig.2.1):

- Robot-like device with 6 DOF provided with 6 drives and the gripper with changeable mock-ups of virtual body and, also, with 6D force-torque sensor measuring vector of effort applied to mock-up with hand;
- Control system for drives of the robot-like device whose SW generates control data providing such mock-up's movement as if it were a real body having the same inertial characteristics.

To verify the second of the mentioned capabilities, i.e. realism of the tactile-kinaesthetic interaction with the imitator of virtual body, dodecahedron was used that should be moved under action of forces applied by hand. Using the computer's software generating movement of virtual body and applied to it forces entering and editing of mass-inertial properties of virtual body were realized (mass, inertia matrix, center-of-mass position). Interaction with bodies of the following mass was studied: 0,05; 0,2; 0,6; 1,2; 3; 4 tons. The inertia matrices were chosen so that with abovementioned masses they would correspond to cubic bodies made out of aluminum. Fig. 5.25 shows the process of man's hand interaction with a body imitator made out of foam plastics.

The experimental results showed that simulation of interaction of hands with bodies is enough realistic. In the first place, it is due to the high frequency of sampling. We succeeded to reduce to a minimum the time expended on measurement of forces and torques applied to a body, calculation of vector of linear and angular velocities and determination for it desired vector of joint coordinates g_d, and, also, delivery of these values to inputs of the servo system controlling drives. It appeared to be 10 ms. Therefore, man perceived the process of interaction as a continuous one.

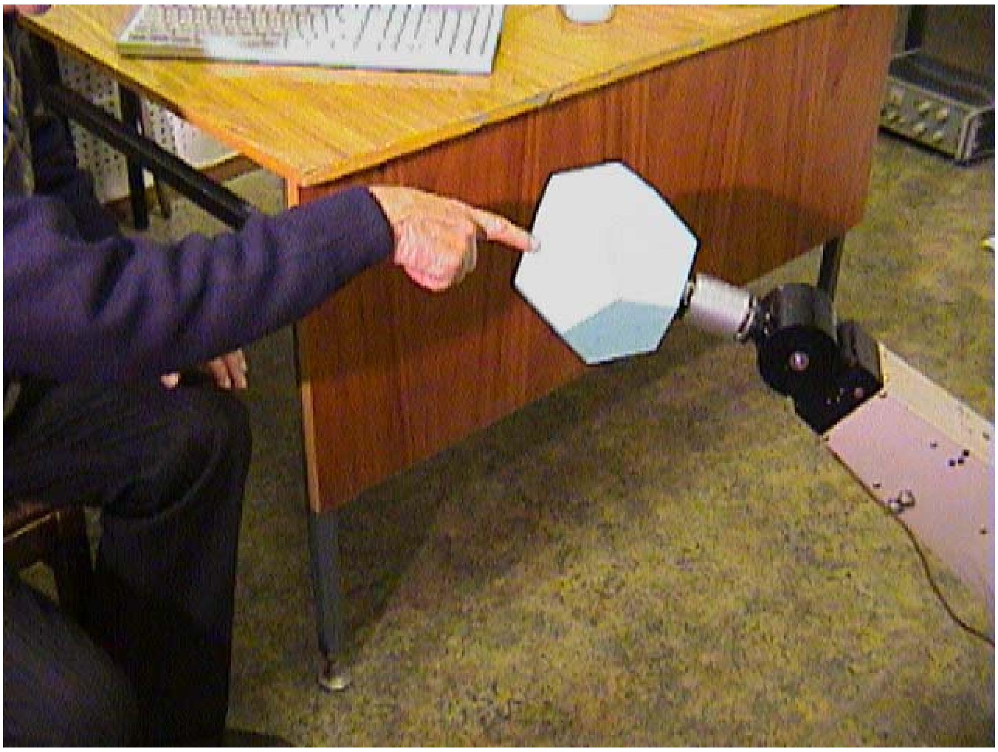


Fig. 5.25 Man's hand interaction with body imitator.

Besides, the error of force interaction, that can be defined as divergence of ideal trajectory of movement of body under applied forces from the real one, measured in the process of experiment, appeared to be insignificant to be felt by man.

The total error of divergence of simulated trajectory of movement increases with reduction of simulated body mass and is equal to 2,5 % of path traveled for time 15 s for a body 1,2 ton and 1,1 % for a body 4 ton (Fig 5.26).

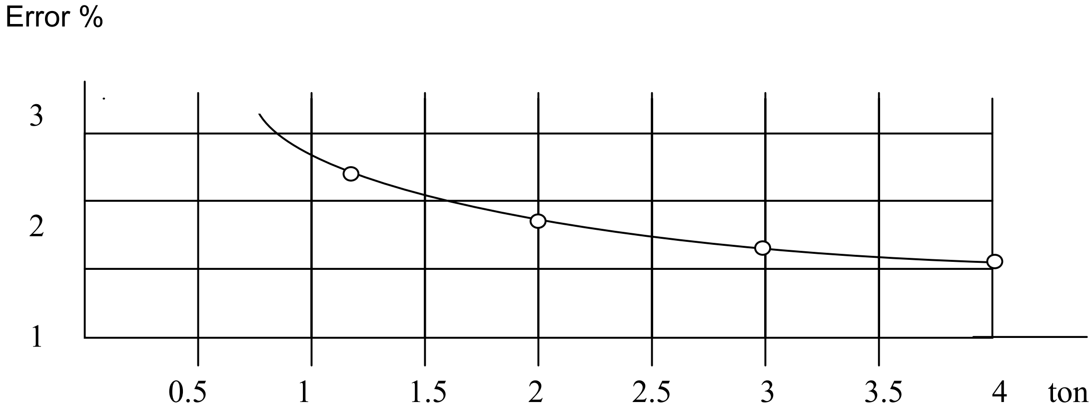


Fig 5.26 Imitation error as function of body mass.

Simulation of movement for bodies of little mass appeared to be difficult owing to its fast going beyond boundaries of free movement because of limited abilities of the manipulator carrying the mock-up resulting from its design.

The abovementioned accuracy of simulation of body movement was measured at short actions of forces, no more than 1,5 s.

In a situation of long time gripping and holding a body the realism of interaction is spoiled by processes caused with dynamic properties of the manipulator. It requires improvement of simulating technique.

In a case when virtual body was a master-slave telecontrolled manipulator the experimental goal was establishing quality of transient processes previous to settling of force applied to the master arm at the manipulator's impacts with hard surface. It was assumed that in this case the applied force is a step function of time. This force must be "copied" on the master arm's handle which is firmly held by man.

The more accurate is the "copying" the more realistic is the kinaesthetic effect of immersion. Ideally, this force is to be, also, the step function identical to that of applied to the virtual manipulator's tool.

Devices which were used for the experiment are integrated in Unit 9 in Fig.2.1.

The master arm's control system is implemented on the base of "Sfera-36" realizing a law described in detail in Report 2 on Task 6, sec.3.2.2 named "Quasi-potential compliant motion control".

The master arm was provided with 6-dimensional wrist force sensor. The stiffness matrix of the sensor is the diagonal matrix $C_w=\{2\cdot10^3, 2\cdot10^3, 2\cdot10^3, 2\cdot10^4, 2\cdot10^4, 2\cdot10^4\}$.

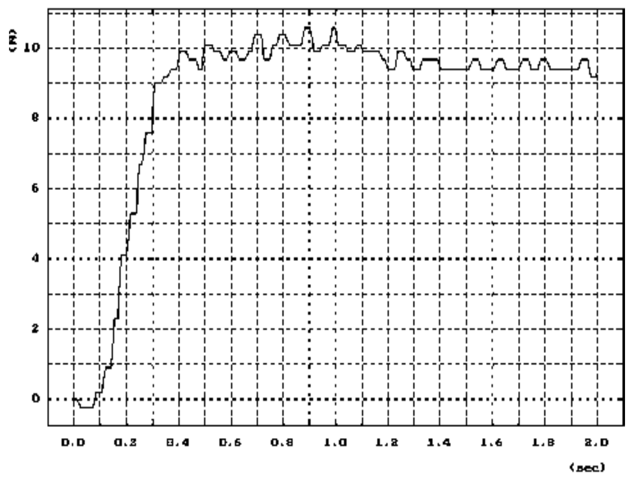
Stiffness of the transmission and links of the master arm is so great that it is possible to consider the transmission and the links as absolutely hard elements in comparison with the wrist of the master-arm.

The constraint equation has form (3.24), where the 1×6 constrained matrix is $K = n = \|0,1,0,0,0,0\|$ the control law is $U=k_1J^{-1}n^Tn(x_e-x_{de}) + k_2J^{-1}(I-n^Tn)(x_{dr}-x_r)$, where x_{dr}=const=0,

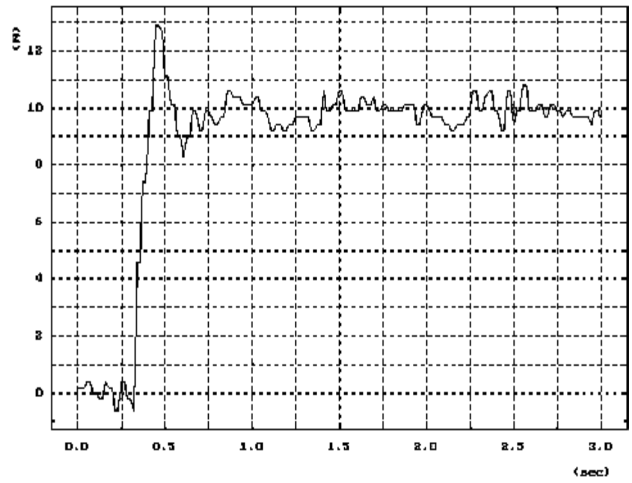
$$x_{de} = L_w^T C_w^T L_w G_d; \qquad G_d = (0,10,0,0,0,0); \qquad x_{de} = (0,10^{-2},0,0,0,0).$$

Figs. 5.27 a,b,c illustrate the transient process, if the contact force for the desired value of G_d is step function. The following values of the gain matrixes $k_1=k_2=10^4$; $k_1=1.5k_2=10^4$, $k_1=0.5k_2=10^4$ are used.

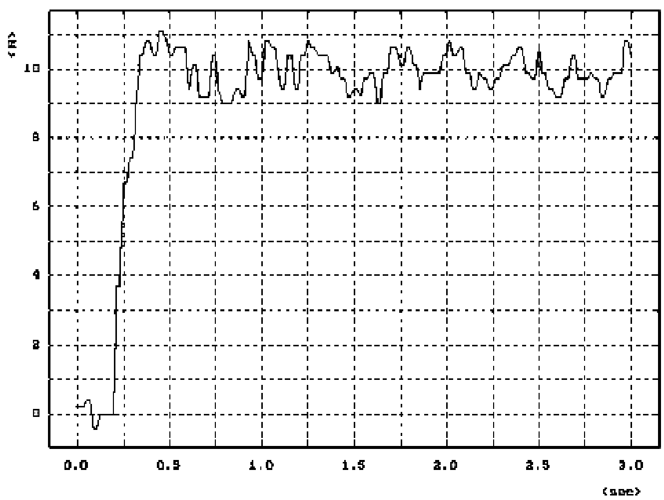
The aim of the second experiment is to test the dynamic accuracy of the contact force during motion of the tool along a complex surface. The velocity vector of the tool has a constant value and it is tangential to the surface. This condition is ensured owing to use of a special method for definition of normal to a surface with the help of the analyses of current contact force *G,* measured by the wrist force sensor.



a)



b)



c)

Fig.5.27. Transient process of the contact force; the desired value G_d=10N:
a) gain matrixes $k_1=k_2=10^4$, b) gain matrixes $k_1=1.5k_2=10^4$, c) gain matrixes $k_1=0.5k_2=10^4$.

The experiments showed enough accurate reproduction on the master arm's handle of forces applied to the virtual manipulator's tool. The attained kinaesthetic effect was absolutely convincing.

Summary

The accomplished studies have resulted in the following:

1. A promising approach was proposed to solving the problem of virtual body's "immersion" in real environment. This approach enables developing the software technology making possible a combination of both visual and tactile-kenaesthetic perception of the "immersion".
2. The proposed approach is the most appropriate for a situation when real scene (in our case mock-up of Orbital Space Station) is beyond man's sight and virtual object is a solid body man's hands are to interact with, or a telecontrolled manipulator.
3. Methods, algorithms and hard-and-software means were developed which realize the proposed approach with less expenses, as compared with the known decisions, providing a more realistic visual and tactile- kinaesthetic effects of perception.
4. A prototype was manufactured of the experimental facility making possible testing and establishing the degree of realism in virtual body's "immersion" in real environment providing visual and tactile- kinaesthetic effects of it.
5. The proposed approach was experimentally verified and showed the following:
   a) with the proposed approach basing on the "Video see-through HMD technology" one easily achieves the standard frame frequency of generated "Augmented Reality" provides the continuity of visual perception making possible considerable sophistication of graphics in our case.
   b) static errors of registaring images of real scene and its computer model is no more than 2-3 pixels approximately 5-8 minutes of arc, providing that the video camera is compensated for nonlinear distortion and thoroughly calibrated for establishing its 6D position with the goal to reduce positioning error to 0,1 mm and to 2-3 minutes of arc and eliminate difference in FOV of real and virtual cameras.
   c) dynamic registration error with the utilized hardware is relatively high: it is 6-8 pixels at head's rotation speed about 2 deg/sec. It was shown that reduction of dynamic error requires utilization of video cameras with short exposition and synchronization of cyclic algorithms' parallel operation at all computers realizing the proposed approach. The synchronization frequency must be equal to the frame frequency.

References

1. Azuma R. "A Motion-Stabilized Outdoor Augmented reality System". IEEE Virtual Reality, 1999.
2. Kuleshov V.S., Lakota N.A. Manipulator control system dynamic. Moscow, Energija, 1971, p.304.
3. Bilateral servosystem designing. Edited by Kuleshov V.S. Moscow, Mashinostroenije, 1980, p.300.
4. Azuma, Ronald T. A Survey of Augmented Reality. Presence: Teleoperator and Virtual Environment 6, 4 (August 1997), 355-385
5. Azuma, Ronald and Gary Bishop. Improving Static and Dynamic Registration in an Optical See-Through HMD. Proceedings of SIGGRAPH'94 (Orlando, FL, 24-29 July 1994), Computer Graphics, Annual Conference Series, 1994, 197-204, + CD-ROM appendix.
6. Kulakov F.M. Augmented of virtual object to real environment. Proc. 5th Int. Conf. on Enterprise Information System. Vol.3, France, April 23-26, 2003, p.p.609-614.
7. "Video-see through Head-mounted Displays for Augmented Reality System", IEEE Virtual Reality, 1999.
8. Kulakov F.M. Robot manipulator control algorithms. Rep. No. JPRS 59717 NTIS, Springfield, Va., Aug 1973.
9. Kulakov F.M., Ignat'ev M.B., Pokrovskiy A.M. The robot-manipulator control algorithms. Joint with Virginia, Joint Publications Research, Service Arlington. 1973.
10. Kulakov F.M. Supervisory Control of Robots - Manipulators, Nauka, Moscow, 1980, 448 p. (In Russian).
11. Kulakov F.M. Methods of Supervisory Control of robot-manipulator, I. Proc. AS USSR, Technical Cybernetics, 1976, N5, p.37-46. (In Russian).
12. Kulakov F.M. Methods of Supervisory Control of robot-manipulator, I. Proc. AS USSR, Technical Cybernetics, 1976, N6, p.78-90.
13. Kulakov F.M. Methods of Supervisory Control of robot-manipulator, I. Proc. AS USSR, Technical Cybernetics, 1977, N1, p.51-66.
14. Kulakov F.M., Trubnikov G.N., Uspensky V.N., Zaitseva T.A. Two channel manipulator grip. Patent N423623 from 02.01.73. (In Russian).
15. Kulakov F.M., Grischkin V.M. A device for reading coordinates from the CRT screen. Patent N1123039 from 08.07.84.
16. Kulakov F.M. Robust flexible robot compliant motion control Moscow,. Proc. AS USSR, Theory and Control Systems, N4, 2000. (In Russian).
17. Anu Rastogi, Paul Milgram, Julius J. Grodski. Augmented Telerobotic Control: a Visual interface for unstructured environments. http://vered.rose.utoronto.ca/people/nu_dir/papers/atc/atcDND.html.
18. B.Brunner, G.Hirzinger, K.Landzettel and J.Heindl. Multisensory Shared Autonomy and Tele-Sensor-Programming-Kay Issues in the Space Robot Technology Experiments ROTEX, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Yokohama, July 23-30, 1993.
19. Anu Rastogi, Paul Milgram, Julius J. Grodski. Augmented Telerobotic Control: a Visual interface for unstructured environments. http://vered.rose.utoronto.ca/people/nu_dir/papers/atc/atcDND.html.
20. Becjzy, A.K. Virtual Reality in Telerobotics. Proc. of the 7th International Conference on Advanced Robotics. Sept. 20-22, 1995 Sant Feliu de Guixols, Catalonia, Spain.

21. Kulakov F., Nechaev A. New Man-Interface for Robotics using Head Tracking System. Proc. Conf. "Science and Computer Technology", Moscow, September 15-17, 2003.
22. Kulakov F. Methodology of Virtual Body Immersion into Real Environment. Proc.4th International Workshop on Computer Science and Information Technologies CSIT'2002, Patras, Greece, 2002.
23. Aceacia, G.M., Callegari, M., Hagemann, D., Michelini, R.C., Molfino, R.M., Pampagnin, S., Razzoli, R.P. and Scwenke, H. Robotic Fixture for Experimenting anthropomorphic vision. Proceedings of the 7th Intern. Conf. on Advanced Robotics. Sept. 20-22, 1995 Sant Feliu de Guixols, Catalonia, Spain, p.p.237-244.
24. Kulakov F.M., Loose H. "Computer simulation of system of hard bodies and application to manipulator". Leningrad, LIIAN, 1986, Preprint pp.52.
25. Solnizev P.I., Kononjuk A.E., Kulakov F.M. "CAD for FMS" Leningrad, "Maschinostroenie", 1990, pp.414.
26. Gantmackher F.R. Theory of Matrices. M.: Nauka, 1966, pp.576.
27. K.S. Fu, R.C. Gonzalez, C.S.G. Lee. *Robotics: Control, Sensing, Vision and Intelligence*. Appendix B2, p.592.